

Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique



uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

ELG 4176 / 4576

COMMUNICATION SYSTEMS

LABORATORY

Fall 2016

Copying of any part of this document is strictly prohibited without written permission.

© P. Galko, 2016

Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique



uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

ELG 4176/4576

COMMUNICATION SYSTEMS

Fall 2016

LABORATORY

General Instructions (“The Rules”):

At the start of the term, the class will divide itself evenly into groups of two students (one group of three may be required). There are only two experiments and one computer based simulation in this laboratory, and these experiments will be conducted in the first two thirds of the term. The schedule of the lab periods for each group (labeled from 1 to 8) is given on the next page. A formal lab report will be required from every student for each of the two labs, while a group report is required for the simulation laboratory. The laboratory portion of the course mark will be derived from the two laboratory reports you submit, the answers to the laboratory preparation questions, your performance during the lab period and the grade for your on the simulation laboratory report. Lab performance is judged on the care and efficiency with which the lab is done, whether you participate in the lab or sit back and let your partners do all the work, your basic competence with lab equipment, and your evident preparedness for the lab. The marks earned for lab performance or for a report are *not* shared by the group members. The lab preparation question answers are graded out of 5 during the lab session, lab performance is graded out of 5, while lab reports are graded out of 40. The experimental lab mark is the sum of these marks for the two experiments, and is 40% of the overall lab mark. The other 60% of the lab mark comes from the simulation laboratory, where all members of the group share the same mark (provided all contribute significantly to the project; should anyone fail to contribute their fair share to the simulation lab and write-up, they will be asked to repeat the lab on their own and their mark will be based only on their later report and will include a penalty).

Everyone is required to have a hard cover laboratory notebook (available in the book store at the Unicenter) in which the experimental observations are entered as well as preliminary analysis of experiment results. The book should be one with graph paper in it so that plots can be made. Any preparation required for a laboratory should also be done in the lab notebook. This work will be quickly checked by the teaching assistant at the start of the lab period and a grade assigned. The same lab book may be used for other electrical engineering courses you are taking. *Anyone who comes to a lab without their own hard cover lab book will not be permitted to participate in that lab session (and will have to make other arrangements to complete the lab on their own at the convenience of the TA; lab performance will be given a grade of zero in all such cases).* At the end of each lab period, your lab book will be signed by the TA to verify the lab results were taken during the lab session. Don't leave the lab until the TA has signed and dated your lab book entries. You may be asked to show your lab book later to verify your results in your lab book as witnessed by the teaching assistant match those in your lab report. (You are never permitted in the lab without the TA being present.)

As the laboratories are largely demonstrations of different signal process notions, reports should include an analysis of the expected results, a comparison with the observed results, and a discussion of any differences. A properly written abstract and a properly worked out set of solutions to the lab preparation questions should also be part of any formal lab report that is submitted. Laboratory reports are due by 11:00 AM one week after the laboratory was scheduled. Reports

should be submitted on regular paper well fastened together (preferably bound in some manner). On each submitted report you must indicate

- i) your name,
- ii) the lab group number, and
- iii) the group members *present in the lab at the time the lab was done*.

Reports must be submitted directly given to a TA or the lab technician (Alain LeHénaff), preferably to the TA who supervised the lab session when the lab was done. Late reports are penalized by deducting 10% of the maximum mark for each day late (i.e., one day late and the best you can get is 90%, two days late and the maximum is 80%, etc.). Late reports may only be submitted directly to the laboratory assistants. Reports will be returned to students approximately one week after the particular laboratory was handed in by the last group to do that lab.

All students must attend their scheduled laboratory sessions and submit a properly prepared lab report on each lab. Failure to attend a session or submit an acceptable formal lab report for the experiment will result in an “incomplete” mark being issued for you in the course. If you are ill, you will have to obtain the appropriate documentation from the University’s Health Services or your physician verifying you were not able to attend the lab for valid reasons, and you will have to make up the lab at some other time (at a time you negotiate with the Teaching Assistants). Labs begin promptly at 8:30 AM (in ELG 4176); all group members must be present from the start. It is unacceptable for anyone to be late for the lab period, and all are expected to fully participate in every lab. Anyone missing for any significant portion of the lab will have to repeat the lab on his/her own at a later date that the TAs will fix for their convenience, not necessarily yours. Your cooperation to ensure the laboratory facilities and time is expected; penalties are imposed when the rules are not followed.

Use of Laboratory Equipment:

The most common source of problems encountered in the labs is equipment failure due to equipment abuse. To minimize the possibility of this occurring, only one student in a group should make the connections in a set-up with the other group members independently checking that the connections are correct BEFORE TURNING ON THE POWER. Pay particular attention to the polarity of power supply connections. Ensure as well that you do not apply a larger signal input than a piece of equipment can handle (in making initial adjustments, always start from a small signal level). Finally, note that ABSOLUTELY no eating, drinking, or smoking is permitted in the laboratory.

ELG 4176 LABORATORY SCHEDULE 2016

Groups Scheduled for Each Lab Session

| Date | Lab | Lab Group(s) |
|---------|-----|--------------|
| Sept. 9 | | |
| 14 | | |
| 21 | | |
| 28 | 1 | 1 |
| Oct. 5 | 1 | 2 |
| 12 | 1 | 3 |
| 19 | 1 | 4 |
| 26 | - | study week |
| Nov. 2 | 1;2 | 5;1,2 |
| 9 | 1;2 | 6;3,4 |
| 16 | 1;2 | 7,8*;5,6 |
| 23 | 2 | 7,8* |
| 30 | | |
| Dec. 7 | | |

*Group 7 and 8 must arrange a time with the TA to do the lab before the scheduled time.



COMMUNICATION SYSTEMS

LABORATORY I

Measurement of Probability Distributions and Correlation Functions

Introduction:

Two of the basic characterisations we find of great importance in analysing signal are the distribution of the values of the signal and the auto- and cross-correlations between signals. In this lab we shall take advantage of a rather unique piece of equipment, the Hewlett-Packard 3721A Correlator, to measure experimentally some of these quantities for some deterministic and random signals.

The 3721A Correlator is an instrument which can sample a signal applied to its input, and sort the sample values it finds into 100 intervals and display the histogram of this result on a CRT screen to give a direct estimate of the distribution over time of a signal's amplitudes (assuming that the sampling instants chosen are representative of arbitrarily chosen points). If a random process that is ergodic is applied to the input, this same procedure will then estimate the process amplitude distribution.

In a similar vein, the 3721 Correlator is also able to sample up to two signals, store the recent sample values and form the sum of the products of one set of samples with differently delayed sets of the other samples so as to form the averages of the $[b(t)$ delayed] products:

$$c_m = \frac{1}{N} \sum_{n=0}^{N-1} a(n\Delta T)b([n-m]\Delta T) \quad \text{for } m \geq 0.$$

This corresponds to the definition of the correlation of two deterministic signals at the various time shifts $(-m\Delta T)$ —only the averages of the product is over a limited time and just for the delay values used. The values that this produces are thus estimates of the crosscorrelation between two deterministic signals or ergodic processes involved at the delay values employed (the first few multiples of the sampling interval). The correlator can display these computed values on a CRT to provide a visual display of the correlation function.

References:

Chapter 5 in John G. Proakis and Masoud Saleh, *Fundamentals of Communication Systems*, Upper Saddle River, NJ: Prentice-Hall, 2005.

Chapter 6 in L.W. Couch II, *Digital and Analog Communication Systems, 6th ed.*, New Jersey, Prentice-Hall, 2001

Chapter 4 in *Communication Systems Engineering, 2nd ed.* by Simon Haykin

Chapters 4 and 5 in *Principles of Communications, 4th ed.* by R.E. Ziemer & W.H. Tranter.

Chapters 10 and 11 in *Modern Digital and Analog Communication Systems, 3rd ed.* by B.P. Lathi.

Preparation:

1. If we were to sample a periodic signal at a randomly chosen point in its period, the value of the sample would be a random variable. The distribution of this random variable is termed the amplitude distribution of the periodic signal. Derive and plot the amplitude distribution of the sinusoid, triangular, and square wave signals (specify both the density function and the distribution function)
2. Calculate and plot the autocorrelation function of a sinusoid and of a 50% duty cycle square wave.
3. Calculate the cross-correlation function between a sinusoid and square wave signal, both of which have the same fundamental frequency.
[Hint: You may find it easier to compute this by making use of the Fourier series of the square wave, noting that the cross-correlation of two sinusoids of different frequency is always zero.]
4. Explain what is meant by the term "bandlimited white noise". Find the autocorrelation function of it and compare it to the autocorrelation function a white noise process.

Apparatus:

- 1 - HP 3721A Correlator
- 1 - LFG-1310 Function generator
- 1 - Wavetek 186 phase-locked generator
- 1 - GR-1383 Noise generator
- 1 - Krohn-Hite 3202 variable filter unit
- 1 - Marconi 2610 true RMS Voltmeter
- 1 - dual channel oscilloscope

The operating instructions and an explanation of the controls of the correlator are given in the Appendix of these lab sheets; READ THESE INSTRUCTIONS BEFORE THE LAB!!

| | | |
|--|------------------|------------------|
| IMPORTANT | IMPORTANT | IMPORTANT |
| The HP 3721A Correlator is a valuable and rare piece of equipment and cannot be replaced any more. Even though the input circuitry is supposed to withstand signals of a 100 V, please make sure that the signals you apply to the input does not exceed the maximum usable input level of 4 V peak-to-peak. | | |

PROCEDURE:

Refer to Item 3-46 and following in the Appendix for instructions on the specifics of different measurement procedures. It should be noted that the averaging switch should be kept in position SUMMATION, the DISPLAY GAIN in position MIN, and the DELAY OFFSET control in the ZERO position. To get meaningful displays, you must stop the sampling process before the highest peak gets to the top of the screen and starts to wrap around.

Part I: Experimental Measurement of a Probability Distribution.

1. Display the probability density function of an asymmetric square wave signal (1 kHz., 2 V peak-to-peak and 25% duty cycle). How is the 25% duty cycle reflected in the display? Note that the display shows the relative densities only, i.e., the top line of the CRT corresponds to the highest value of the density function. Try different duty cycles and verify the results. Also change the voltage level and check the horizontal scale calibration.
2. Switch the FUNCTION switch to INTEGRAL and measure the (cumulative) probability distribution functions for the signals in part 1. Do the displays agree with the theory? Notice that the bottom line and the top line represent probability 0 and 1 respectively.
3. Measure both the probability density function and the probability distribution function for:
 - a) a sinusoidal signal
 - b) a triangular signal
 Do the displays agree with the results obtained in the preparation?

4. Display the probability density function of the GR-1383 noise generator. What form does this probability density function seem to have? Estimate its mean value and its variance. Set up a KROHN-HITE filter as a band pass filter, and connect this filter between the noise generator and the HP 3721A correlator. Choose a number of bandwidths—e.g., 100 kHz, 500 kHz and 800 kHz, with a centre frequency of 1 MHz. Display the probability density function of each of the resulting bandpass noises. Comment on its shape. Estimate their mean and variance.

Part II: Measurement of Correlation functions.

1. Apply a 2 V peak-to-peak 2 kHz sinusoidal, triangular and square wave in turn to the correlator unit and display the autocorrelation function of each of these signals. How does the period of the autocorrelation function compare with the signal period? Compare your experimental results with those of the preparation.
2. Apply a 1 V peak-to-peak, 2 kHz sinusoid to one input of the correlator and a 1 V peak-to-peak square wave from the Wavetek phase locked generator (locked to the same frequency as the sinusoid) to the other input of the correlator. Measure the correlation between the two signals. Vary the phase of one signal relative to the other and observe the effect of the cross-correlation..
3. Replace the square-wave with the output of the noise generator and measure the correlation between the sinusoid and the noise signal.

APPENDIX

HP 3721A Correlator (extracts from Operating Manual)

Section 1

INTRODUCTION

1-1 DESCRIPTION

1-2 The Hewlett-Packard Model 3721A Correlator is a compact, digital instrument capable of computing and displaying, in real time, autocorrelation, crosscorrelation and probability functions. It has the added facility for recovery of repeated events buried in noise frequently referred to as 'signal averaging'. To avoid by any confusion with arithmetic averaging, an essential part of any statistical measurement, this process will be referred to as Signal Recovery when applied to the Correlator.

1-3 In this manual, the instrument will be referred to as the 3721A or Correlator.

1-4 FEATURES

1-5 The 3721A features:

- a. Simultaneous computation and display of 100 points of the function selected. The computed function may be displayed indefinitely without deterioration.
- b. Time interval between points on the display selectable from 1 μ s/mm to 1 s/mm: in correlation this represents a delay span from 100 μ s to 100 s. By using an external clock the interval can be extended to any delay increment greater than 1 μ s
- c. A choice of two methods of averaging: SUMMATION AVERAGING—the digital equivalent of pure integration or EXPONENTIAL AVERAGING—the digital equivalent of analog exponential (RC) smoothing. The averaging time constant can be varied from 36 ms to over 150 days.
- d. Quick-look analysis feature giving rapid indication of the final value of the function—in exponential averaging modes.
- e. Outputs for transferring displayed data to external X-Y recorder and oscilloscope.
- f. Compatibility with *hp* Model 3722A Noise Generator—making a powerful combination for dynamic response measurements.
- g. Data Interface Option Series providing direct interfacing of the 3721A with a computer or tape punch.

h. Delay Offset Option Series enabling the instrument to be used with greater resolution, up to a maximum of 1 point in 1150.

1-6 FUNCTIONS COMPUTED

1-7 Correlation. The 3721A computes and displays the following correlation functions:

Autocorrelation of either Channel A or B input.

Crosscorrelation between Channel A and B; A delayed with respect to B.

Crosscorrelation between Channel A and B; B delayed with respect to A.

1-8 The instrument performs simultaneous computation and display of the correlation function for 100 values of delay. The vertical calibration (V^2/cm) is automatically displayed on an illuminated panel.

1-9 Signal Recovery. The 3721A improves the signal-to-noise ratio of repeated events, provided each event is marked by a synchronising pulse. After each synchronising pulse, a series of 100 samples of channel B input is taken and averaged with the corresponding samples from previous series. The vertical calibration (V/cm) is automatically displayed on the illuminated panel.

1-10 Probability. The 3721A computes and displays the following probability functions:

The amplitude probability density function (pdf) .

The integral of the pdf, the cumulative amplitude probability distribution function (cdf) .

1-11 Both are performed on Channel A signal input only. The signal's amplitude is displayed horizontally, with zero volts in the centre, and vertical deflection represents the probability (either density or integral).

1-14 Delay Offset (*An Option*) Enables the user to introduce a selected amount of precomputational delay into the delayed channel, to view the significant part of the computed function with greater resolution: operational in correlation modes only. There are four alternatives available, giving maximum precomputational delays of 150, 250, 450 or $1050\Delta t$ throughout the range.

Table 1-1 Correlator Specifications

INPUT CHARACTERISTICS

Two separate input channels, A and B, with identical amplifiers.

Input amplifier bandwidth. DC to 250 kHz nominal. Lower cut-off frequency selectable, dc or 1-Hz (10% down at 3 Hz).

Input range. Signals accepted from 40 mV to 4 V rms, over 6 ranges.

Overload. Maximum permissible voltage at input: DC coupled 120 V peak; ac coupled 400 V = dc + peak ac.

Analog-to-digital conversion. Fine quantizer: 7 bits. Coarse quantizer (feeds delayed channel): 3 bits. Coarse quantizer linearized by internally-generated wideband noise (dither).

Input impedance. Nominally 1 M Ω , shunted by 100 pF to ground

CORRELATION MODE

Computes the following functions:

- Autocorrelation of A input
- Autocorrelation of B input
- Crosscorrelation of A and B inputs, A delayed
- Crosscorrelation of A and B inputs, B delayed

Simultaneous computation and display of 100 values of auto or crosscorrelation function. Display sensitivity indicated directly in V²/cm on illuminated panel. Non-destructive read-out; computed function can be displayed for an unlimited period without deterioration. (Non-permanent storage; data cleared on switch-off.)

Timescale. (TIME/MM = delay increment Δt) 1 μ s to 1 second (total delay span 100 μ s to 100 seconds) in 1, 3.33, 10 sequence with internal clock. Other delay increments with external clock; minimum increment 1 μ s (1 MHz), no upper limit.

Delay offset. Option Series 01 provides delay offset (precomputational delay) facility. Enables display resolution to be increased to magnify area of interest (Auto and Crosscorrelation measurements only).

Display sensitivity. 5 $\times 10^{-6}$ to 5 V²/cm. Vertical calibration automatically displayed by illuminated panel.

Vertical resolution. Depends on display sensitivity. Minimum resolution is 25 levels/cm. Interpolation facility connects points on display.

Averaging. Two modes are provided: Summation (true integration) or Exponential (digital 'RC' averaging).

Summation mode. Computation automatically stopped after N process cycles, at which time each point on the display represents the average of N products. N is selectable from 128 to 128 \times 1024 (2⁷ to 2¹⁷ in binary steps). Display calibration automatically normalized for all values of N.

Exponential mode. Digital equivalent of RC averaging [passing the signal product through an RC lowpass filter], with time constant selectable from 36 ms to over 10⁷ seconds. Approximate time constant indicated by illuminated panel. Display correctly calibrated at all times during the averaging process

PROBABILITY MODE (Channel A only)

Displays either amplitude probability density function (pdf) or integral of the pdf of channel A input. Signal amplitude represented by horizontal displacement on display, with zero volts at centre; vertical displacement represents amplitude probability.

Display sensitivity. Horizontal sensitivity 0.05 to 2 V/cm in 5, 10, 20 sequence.

Horizontal resolution. 100 discrete levels in 10 cm wide display = 10 levels/cm.

Vertical resolution. 256 discrete levels in 8 cm high display = 32 levels/cm.

Vertical calibration

Summation averaging. Process automatically stopped when any one point of the display has occurred approximately N times: N being selectable from 128 to 131,072 (2⁷ to 2¹⁷ in binary steps). With the DISPLAY GAIN switch set to MIN, this corresponds to 8 cm vertical deflection. The total number of occurrences of the signal at all amplitudes may be obtained from a counter connected to the rear-panel PROCESS CLOCK output.

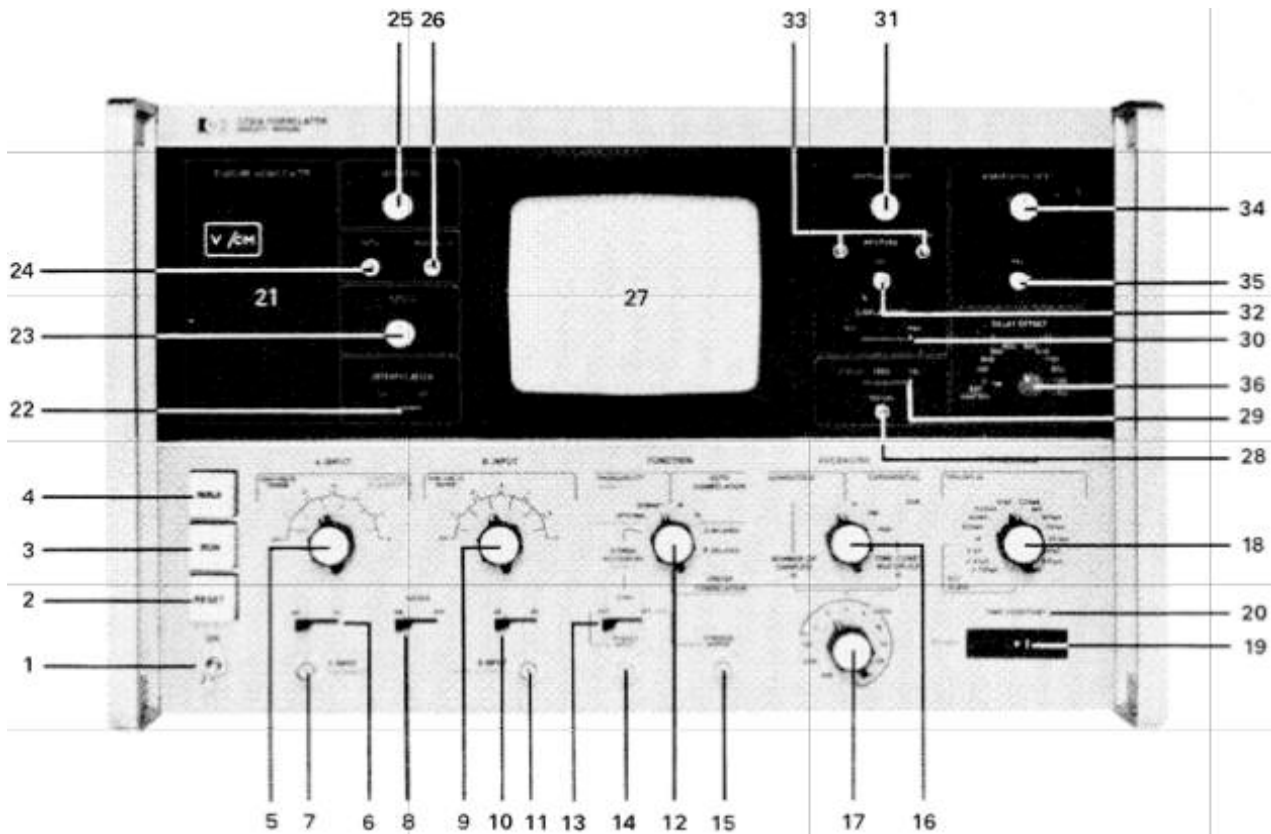
Exponential averaging. Continuous updating of display, with time constant as given for Correlation and Signal Recovery modes. The Correlator is not vertically calibrated in exponential mode.

Sampling rate. 1 Hz to 3 kHz in 1, 3, 10 sequence with internal clock. Other sampling rates with external clock; maximum frequency 3 kHz, no lower frequency limit.

Section 3

OPERATING INSTRUCTIONS

Figure 3-1 Front panel controls, connectors and indicators



- | | |
|--|--|
| <p>1 LINE ON: toggle switch. Controls ac supply to the instrument.</p> <p>2 RESET: push-button control. Clears stores and pre-sets registers and counters to their initial states. Glows when operating.</p> <p>3 RUN: push-button control. Starts Correlator processing. Glows when operating.</p> <p>4 HOLD: push-button control. Stops processing but retains display information already processed. Glows when operating. The Correlator goes into the HOLD state automatically when any of the major front-panel controls are operated.</p> <p>5 A INPUT RMS VOLTS RANGE: rotary switch. Sets gain of channel A amplifier. In Probability modes the inner (red) scale provides calibration for the horizontal axis of the display.</p> <p>6 A INPUT AC/DC: lever switch. Determines signal coupling to channel A amplifier. Lower ac cut-off frequency 1 Hz.</p> <p>7 A INPUT/PROBABILITY: BNC connector. Signal input connector for channel A and Probability measurements.</p> <p>8 DITHER ON/OFF: lever switch. Introduces wideband Gaussian noise into the delayed channel; improves the analog-to-digital conversion linearity for non-Gaussian signals</p> <p>9 B INPUT RMS VOLTS RANGE: rotary switch. Sets gain of channel B amplifier.</p> | <p>10 B INPUT AC/DC: lever switch. Determines signal coupling to channel B amplifier. Lower ac cut-off frequency 1 Hz.</p> <p>11 B INPUT/SIGNAL RECOVERY: BNC connector. Signal input connector for channel B and Signal Recovery measurements.</p> <p>12 FUNCTION: rotary switch. Selects measurement required: PROBABILITY A INPUT only; DENSITY or INTEGRAL AUTOCORRELATION; A or B CORRELATION; A DELAYED OR B DELAYED SIGNAL RECOVERY B INPUT only.</p> <p>13 SYNC EXT/INT: lever switch. Operational in Signal Recovery mode only. Selects trigger source (EXTERNAL or INTERNAL) for initiating averaging sweep.</p> <p>14 TRIGGER INPUT: BNC connector. Input for external synchronising pulse to initiate Signal Recovery averaging sweep. Operational in Signal Recovery mode only.</p> <p>15 STIMULUS OUTPUT: BNC connector. Provides output pulse to excite external event being recovered. Operational in Signal Recovery mode only.</p> <p>16 AVERAGING: rotary switch. Selects method of averaging—SUMMATION or EXPONENTIAL.</p> <p>17 NUMBER OF SAMPLES N/TIME CONST MULTIPLIER N: rotary switch. In SUMMATION averaging mode selects Number of Samples; in EXPONENTIAL selects the Time Constant Multiplier.</p> |
|--|--|

- 18 **TIMESCALE TIME/MM Δ :** rotary switch. Sets delayed sampling interval for Auto and Crosscorrelation modes and the sampling interval for the Signal Recovery and Probability modes. Provides calibration of the horizontal axis of the display in modes other than Probability.
- 19 **TIME CONSTANT:** illuminated panel indicator. Enables experiment time (Summation) or averaging time constant (Exponential) to be estimated.
- 20 **TIME CONSTANT:** signal lamp. When lit, lamp indicates experiment complete (Summation) or averaging time constant reached (Exponential).
- 21 **DISPLAY SENSITIVITY:** illuminated panel indicator. Provides calibration of the vertical axis of the display in modes other than Probability.
- 22 **INTERPOLATION ON/OFF:** lever switch. In the ON position, adjacent points of the display are interconnected.
- 23 **FOCUS:** potentiometer. Controls sharpness of CRT beam.
- 24 **ASTIG:** potentiometer. Adjusts geometry of CRT beam.
- 25 **INTENSITY:** potentiometer. Controls brightness of CRT beam.
- 26 **TRACE ALIGN:** potentiometer. Rotates trace around centre of CRT screen.
- 27 **DISPLAY:** 10 cm wide by 8 cm high Cathode Ray Tube with graticule calibrated in 1 cm squares. Major axes calibrated in 2 mm graduations.
- 28 **RECORD:** push-button control. Initiates a single sweep at a speed suitable for X-Y recorder.
- 29 **DISPLAY/ZERO/CAL:** lever switch. Left in DISPLAY position for normal operation. ZERO and CAL positions provide calibration points in centre and lower left hand corner of display respectively for X-Y recording.
- 30 **DISPLAY GAIN MIN/I/MAX:** lever switch. Provides three magnifications of the vertical axis of displayed trace.
- 31 **VERTICAL SHIFT:** potentiometer. Moves trace vertically on the CRT screen.
- 32 **CAL:** potentiometer, Calibrates vertical axis of display.
- 33 **RESTORE UP/DOWN:** push-button switches. Enable trace to be moved vertically on the CRT screen for investigation of points on the trace overrunning the screen, without disturbing the setting of the VERTICAL SHIFT control (31).
- 34 **HORIZONTAL SHIFT:** potentiometer. Moves trace horizontally on the CRT.
- 35 **CAL:** potentiometer. Calibrates horizontal axis of display.
- 36 **DELAY OFFSET (if Option Series 01 fitted):** rotary switch. Not fitted to standard 3721A. See Section IV of this manual for more details.

3-1 INTRODUCTION

3-2 This section contains information and instructions for the operation of the 3721A Correlator. Paragraphs 3-3 through 3-45 give information on preliminary setting up procedures and Paragraphs 3-46 through 3-50 give detailed step-by-step measurement procedures.

3-3 SETTING CRT CONTROLS

CAUTION

BEFORE CONNECTING THE CORRELATOR TO THE AC POWER SUPPLY BE SURE THAT THE CORRECT VOLTAGE IS SELECTED AND THAT THE FUSE IS CORRECTLY RATED

3-4 Set the CRT controls, which are similar to those of a conventional oscilloscope, as follows:

1. Connect the instrument to the correct power supply and switch on. Allow 15 minutes for warm-up.
2. Make sure that the instrument is in the RESET condition.
3. Set DISPLAY/ZERO/CAL switch to ZERO.
4. Adjust INTENSITY, VERTICAL SHIFT and HORIZONTAL SHIFT controls for a spot trace of suitable brightness near the centre of the screen.
5. Adjust FOCUS and ASTIG controls for a clear circular spot.
6. Adjust VERTICAL SHIFT and HORIZONTAL SHIFT controls to place the spot on the screen centre.
7. Set AVERAGING switch to SUMMATION.
8. Set DISPLAY GAIN switch to MIN.
9. Set DISPLAY/ZERO/CAL switch to DISPLAY. Readjust INTENSITY control for normal trace brightness.
10. Adjust TRACE ALIGN control for trace parallel to the horizontal axis.
11. Adjust HORIZONTAL CAL control for 9.9 cm trace width, starting at left hand origin.
12. Set FUNCTION switch to PROBABILITY INTEGRAL.
13. Set TIMESCALE switch to 333 μ S.
14. Set NUMBER OF SAMPLES N switch to 1 \neq 1024.
15. Press RESET and then RUN push-buttons.

16. The trace will split, one half rising to the top of the graticule and the other half remaining on the bottom. If necessary, adjust VERTICAL SHIFT and CAL controls for correct display.

3-6 Local Control. The three push-button controls operate as follows:

- a. RUN sets the Correlator into the processing state.
- b. HOLD stops the processing at the end of the current updating cycle. On pressing the RUN push-button, processing restarts. Information already in the delay store is not cleared on restart. Operation of any of the major front-panel controls automatically sets the HOLD condition.
- c. RESET clears all information from the stores. The Correlator is in the RESET condition when it is switched on.

3-8 INPUT SIGNALS

3-9 The Correlator accepts signal frequencies from 0 (dc) to 250 Hz. If the signal contains frequencies of about 1 Hz or lower, set the AC/DC coupling switch(es) to DC; otherwise to AC. For normal use of the analog-digital converters, input signals should be Gaussian and have amplitudes lying within the range 40 mV to 4 V rms. If the signal is non-Gaussian the DITHER switch should be set to ON.

3-10 If the level of the signal to be investigated is unknown it is necessary to measure it using a suitable voltmeter (e.g., hp 3400A) and set the INPUT range switch(es) accordingly. If the level is above or below the range of the 3721A it will be necessary to insert suitable attenuation or pre-amplification.

3-11 CRT DISPLAY

3-12 Functions computed by the Correlator are displayed on the internal CRT. Calibration of the screen varies with the function being displayed.

3-13 Signal Recovery and Correlation Calibration. In Signal Recovery and all Correlation modes horizontal calibration is in TIME/MM and may be read directly off the TIMESCALE switch scale, whilst vertical calibration is automatically given on the illuminated DISPLAY SENSITIVITY† panel to the left of the CRT.

†This calibration is correct for all combinations of switches affecting the vertical calibration but, because there are many combinations it has been found necessary to minimize the ranges used and to adjust the scale of the

CRT display in other cases. This means that although normally the full-scale vertical display can occupy 8 cm (full screen height), occasionally it will occupy 10 cm (2 cm more than full screen height) or only 64 cm (less than full screen).

In probability modes the DISPLAY SENSITIVITY indicator is inoperative and the full-scale vertical display can always occupy 8 cm, irrespective of switch combinations.

3-14 Probability Calibration. In PROBABILITY modes horizontal calibration is in V/CM and may be read off the A INPUT RMS VOLTS RANGE switch scale (red scale). Vertical calibration varies with the mode selected (DENSITY or INTEGRAL) and with the method of AVERAGING.

3-15 With EXPONENTIAL averaging the vertical axis is uncalibrated.

3-16 With SUMMATION averaging and PROBABILITY INTEGRAL mode the display freezes when the input signal has been sampled 4N* times. (See Table 3-2 for difference between N and N*.) Calibration is automatic in that the highest point on the screen represents 100% probability and the lowest point represents zero probability. With the DISPLAY GAIN switch set at MIN, these are top and bottom of the CRT screen respectively.

3-17 With SUMMATION averaging and PROBABILITY DENSITY mode the display freezes when N* counts have been made in the most measured amplitude interval (see Table 3-2 for difference between N and N*).

3-18 For calibration the display may be normalized; this can be accomplished by using the procedure detailed below.

- i. Measure the rms value of the signal using a suitable voltmeter, and let this value equal q.

Table 3-2
Corresponding values of N and N*

| Setting of NUMBER OF SAMPLES N Switch | Value of N | Value of N* |
|---------------------------------------|------------|-------------|
| 128 | 128 | 127 |
| 256 | 256 | 254 |
| 512 | 512 | 508 |
| 1×1024 | 1024 | 1016 |
| 2×1024 | 2048 | 2032 |
| 4×1024 | 4096 | 4064 |
| 8×1024 | 8192 | 8128 |
| 16×1024 | 16384 | 16256 |
| 32×1024 | 32768 | 32512 |
| 64×1024 | 65536 | 65024 |
| 128×1024 | 131072 | 130048 |

- ii. Connect the signal to A INPUT/PROBABILITY connector and set A INPUT RMS VOLTS RANGE switch accordingly.
- iii. Set 3721A FUNCTION switch to PROBABILITY DENSITY, AVERAGING switch to SUMMATION and DISPLAY GAIN switch to MIN.
- iv. Connect rear-panel PROCESS CLOCK†† output connector to a suitable counter (e.g., hp 5245L). ††Instruments fitted with an option from Series 01 have a PROCESS CLOCK output that does not stop after processing is complete. Instead, use Pin 19 of the 50-way connector on the rear of the Correlator as an external stop signal for the Counter. This pin at 12 V whilst the time constant lamp is not lit and falls to 0 V when it lights.
- v. Set the counter to zero and press RESET push-button on 3721A.
- vi. Press RUN push-button. When one point of the display reaches 8 cm deflection (the top of the CRT

screen), processing stops. The reading on the counter is the number of process cycles; let this equal A.

- vii. The probability at the highest point of the display is:

$$p(x)_{\text{peak}} = EQ \sqrt{F(N^*, A)}$$

where N* is the number of samples in the voltage window where the highest point lies. N* is derived from the setting of the NUMBER OF SAMPLES switch by reference to Table 3-2.

- viii. The normalized probability at the highest point $EQ \sqrt{p(x)_{\text{peak}}}$ is given by:

$$EQ \sqrt{p(x)_{\text{peak}}} = EQ \sqrt{F(N^* q, A w)}$$

where w is the voltage window at which the highest point lies.

$$w = \frac{(\text{HORIZ V/CM setting})}{10}$$

- ix. With one point calibrated, all other points on the plot can be calibrated in proportion. For example, if one point is y cm from the baseline and x cm from the centre line, the normalized coordinates are:

$$\bar{p}(x)_{\text{peak}} = \frac{y}{8} \bar{p}(x)_{\text{peak}}$$

$$\bar{x} = \frac{x(\text{HORIZ V/CM setting})}{q}$$

3-25 SAMPLING RATE

3-26 The choice of the TIMESCALE control setting in Correlation and Signal Recovery modes is important if maximum statistical information is to be recovered. Firstly, in these modes the choice of the sampling interval must comply with the requirements of the Sampling Theorem which states that, for complete recovery of the statistics of a signal, the rate at which it is sampled be at least twice the highest significant frequency present in the signal. Secondly, if the CRT display is being used as the output device, erroneous results may be interpreted due to the inability of the eye to interpolate in certain circumstances. This can happen with a periodic signal when there are less than about five display points per cycle.

3-27 Figure 3-3 shows the autocorrelation function of a sine wave where the number of points per cycle is more than two (thus fulfilling the requirements of the Sampling Theorem).

3-28 However the eye tends to see the trace shown solid rather than the correct dashed one. If the TIMESCALE setting is decreased the correct trace becomes readily apparent and the possibility of false interpretation no longer exists (see Figure 3-4)

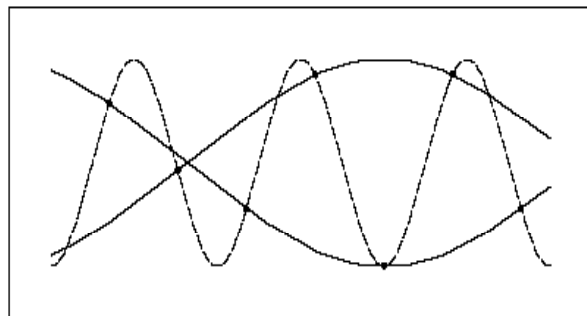


Figure 3-3 Autocorrelation function of a sine wave

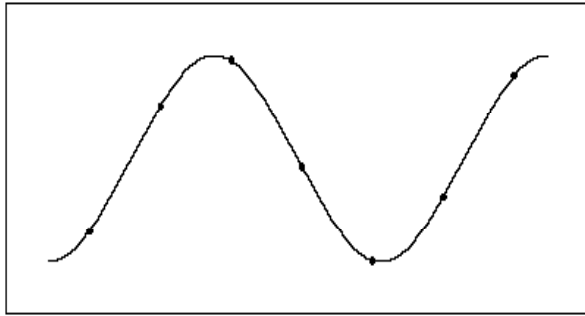


Figure 3-4 The autocorrelation function with the timescale setting decreased

3-29 Note that the appearance of the trace is purely a subjective matter and that the accuracy and usefulness of any calculations are not affected provided the correct information to be used is appreciated.

3-30 This effect, sometimes termed "Optical Aliasing", must not be confused with true aliasing, which occurs when the Sampling Theorem requirements are not met. Optical aliasing may occur both in the presence, and absence, of true aliasing, and providing the sampling requirements are met, may always be removed by decreasing the sample interval. Use of the INTERPOLATION facility offered in the 3721A will often help in detecting Optical Aliasing, and in general display interpretation.

3-31 AVERAGING

3-32 Two methods of averaging are provided on the Correlator; SUMMATION and EXPONENTIAL.

3-33 Summation. When making Auto and Crosscorrelation measurements or performing Signal Recovery experiments using Summation averaging the computation stops automatically after N updatings of the displayed function. N, the number of samples taken, is set by the front panel selector switch in the range 128 to 128 x 1024 (27 to 217) in binary steps.

3-34 Completion of the number of samples, N, being taken, is indicated when the TIME CONSTANT lamp lights: a further 100 samples are then taken for complete updating of all 100 displayed points.

3-35 In Probability Density measurements the processing is arranged such that when the highest point has occurred N* times processing ceases. For convenience it is arranged that this point occupies 8 cm (full screen height). (See Table 3-2 for difference between N and N*.)

3-36 In Probability Integral measurements processing ceases after 4 N* samples have been taken. Again for convenience the highest point occupies 8 cm (full-screen).

3-37 Exponential. Exponential averaging on the 3721A is the digital equivalent of resistance capacitance averaging where the time constant may be varied. This averaging time constant is the product of N x Δt x number displayed on the illuminated TIME CONSTANT panel. (See Paragraph 3-40).

3-38 With Exponential averaging in the 3721A this time constant is progressively increased, automatically, from a low value at the start of the experiment, to the value given by the front panel TIME CONSTANT indicator. The CRT display is calibrated throughout this progression giving the Operator a 'quick-look' facility, from which he can gain useful information about the signals under investigation, without having to wait until the final time constant is reached.

3-39 The TIME CONSTANT lamp will light when the final value of time constant (as indicated by the TIME CONSTANT indicator) has been reached, but processing will continue indefinitely, until stopped by the Operator.

3-40 CALCULATION OF TIME CONSTANTS AND SUMMATION TIME

3-41 The estimation of the variance or statistical accuracy likely in analyzing a signal using the 3721A involves the calculation of suitable Experiment Times (in Summation averaging) or Time Constants (in Exponential averaging) to obtain the desired accuracy and the relationship of the TIME CONSTANT lamp to these. Table 3-3 below sets out the calculations involved for various combinations. The Averaging Time Constant Calculator (hp part number 03721-95011) supplied with the instrument, aids in calculating these results.

NOTE: Δt is the setting of the TIMESCALE switch and N is the setting of the NUMBER OF SAMPLES/TIME CONST MULTIPLIER switch.

3-42 Signal Recovery. The process of Signal Recovery is a method of improving the signal-to-noise ratio in the observation of a repeated event. The improvement is in proportion to the square root of N (as set by the NUMBER OF SAMPLES/TIME CONST MULTIPLIER switch) and therefore the length of any experiment is dependent on the signal-to-noise ratio improvement required and on the time between the trigger pulses denoting the start of each repeated event.

3-43 With Summation averaging the experiment time will be equal to:

$$(N \times \text{time interval between trigger pulses})$$

3-44 With Exponential averaging the signal is in effect, smoothed by a resistance capacitance filter of a preselected time constant equal to:

$$(N \times \text{time interval between trigger pulses})$$

3-45 Consider this time interval for the two cases of external and internal trigger pulses.

a. External Trigger signal. With both methods of averaging the time interval between trigger pulses should be at least [100 Δt + (200 x 10⁶)] seconds if all repetitions are to be averaged; if less than this the instrument will function, but less efficiently, as some repetitions will be ignored.

b. Internal Trigger (Stimulus) signal. With both methods of averaging the time interval between trigger pulses is [100 Δt + (200 x 10⁶)] seconds.

Table 3-3 Calculation of Time Constants and Experiment Times

| $\Delta t \geq 333\mu s$ (Normal mode) | $\Delta t = 100\mu s$ or $33\mu s$ (10:1 Batch mode) | $\Delta t = 10\mu s, 3.33\mu s$ or $1\mu s$ (100:1 Batch mode) |
|--|---|---|
| AUTOCORRELATION/CROSSCORRELATION | | |
| Summation: The TIME CONSTANT lamp will light N Δt s after the RUN push-button is pressed. This is the value to be used in the calculation† of both variance and experiment times. | Summation: The TIME CONSTANT lamp will light 10.N.Δt after the RUN push-button is pressed. This is the value to be used in the calculation† of experiment times. For variance calculations† the product N Δt must be used. | Summation: Approximate time for the TIME CONSTANT lamp to light is given by the Calculator and is used for estimating experiment times. The product N.Δt is to be used for variance calculations†. |

†Note: A further $100\Delta t$ must be included in these calculations, to allow for the final updating of all 100 points.

| | | |
|--|---|--|
| <p>Exponential: The averaging time constant is $N\Delta t$ times the white illuminated factor in the TIME CONSTANT indicator panel. This is the value to be used in the calculation of both variance and experiment times.</p> | <p>Exponential: The averaging time constant is given by the Calculator. The value to be used in variance calculations is the product of $N\Delta t$ and the white illuminated factor, (Note that the green illuminated factor should be ignored in these calculations.)</p> | <p>Exponential: The averaging time constant is given by the Calculator. The product of $N\Delta t$ and the white section of the illuminated panel is to be used for variance calculations.</p> |
| PROBABILITY DENSITY | | |
| <p>Summation: Processing time is a function of the characteristics of the signal being measured. Without prior knowledge of the amplitude probability of the signal, it is not possible to predict experiment times. The time for variance calculations is the same as the experiment time.</p> | <p>Sampling set at $333\mu s$ —calculations as for Normal mode.</p> | <p>Sampling set at $333\mu s$ —calculations as for Normal mode.</p> |
| <p>Exponential: The averaging time constant for calculating both experiment times and variance is given by the product of $N\Delta t$ and the white section of the illuminated panel.</p> | | |
| PROBABILITY INTEGRAL | | |
| <p>Summation: The TIME CONSTANT lamp will light $4.N^* \Delta t$ after the RUN push-button is pressed. This value is used to predict experiment times and for variance calculations. NOTE: for the significance of the expression N^* please refer to Table 3-2.</p> | <p>Sampling set at $333\mu s$ —calculations as for Normal mode.</p> | <p>Sampling set at $333\mu s$ —calculations as for Normal mode.</p> |
| <p>Exponential: The averaging time constant is $N\Delta t$ times the white section of the illuminated panel and this is the value used to predict experiment times and for variance calculations.</p> | | |

3-46 MEASUREMENT PROCEDURES

3-47 Autocorrelation. Autocorrelation gives a measure of the similarity between a signal and a delayed version of itself, expressed as a function of the delay. Autocorrelation measurements can be made on either input A or input B. Input A measurement procedure is as follows, (for channel B measurements set appropriate B INPUT controls and turn the FUNCTION switch to AUTOCORRELATION B):

1. Set AC/DC switch for coupling required. When the input signal is dc or has frequency components less than approximately 1 Hz, the DC position must be selected.
2. Set A INPUT RMS VOLTS RANGE switch for level of the signal being investigated (see Paragraph 3-8).
3. If the signal is Gaussian, set DITHER switch to OFF, otherwise to ON.
4. Set FUNCTION switch to AUTOCORRELATION A.
5. Set AVERAGING switch to SUMMATION or EXPONENTIAL as required.
6. Set TIMESCALE switch for correct sampling rate for signal being measured. Sampling rate must be at least twice the highest significant frequency component of the input signal. Horizontal calibration of the display given directly from the setting of the TIMESCALE control in TIME/MM.
7. Set NUMBER OF SAMPLES N/TIME CONST MULTIPLIER N switch as required.
8. Connect input signal to A INPUT connector.
9. Press RESET push-button.
10. Press RUN push-button.
11. Allow processing to continue for a suitable length of time. The TIME CONSTANT indicator and lamp enable approximate measurement times to be calculated. (See Paragraph 3-40).
12. Vertical calibration of the display is shown on the DISPLAY SENSITIVITY panel in V^2/CM .

3-48 Crosscorrelation. Crosscorrelation gives a measure of the degree of similarity between two signals, expressed as a function of the time shift between them. Crosscorrelation measurements can be made with either input A or input B delayed. Measurement procedure is as follows:

1. Set AC/DC switches for A INPUT and B INPUT couplings required. When an input signal is dc or has frequency components less than approximately 1 Hz, the DC position must be selected.
2. Set A INPUT RMS VOLTS RANGE and B INPUT RMS VOLTS RANGE switches for levels of the signals-being investigated (see Paragraph 3-8).

3. If the signal connected to the delayed channel (coarse quantizer) is Gaussian, set DITHER switch to OFF, otherwise to ON.
4. Set FUNCTION switch to CROSSCORRELATION; A DELAYED or B DELAYED as required.
5. Set AVERAGING switch to SUMMATION or EXPONENTIAL as required.
6. Set TIMESCALE switch for correct sampling rate for signals being measured. Sampling rate must be at least twice the highest significant frequency of the higher frequency input signal.
Horizontal calibration of the display is given directly from the setting of the TIMESCALE control in TIME/MM.
7. Set NUMBER OF SAMPLES N/TIME CONST MULTIPLIER N switch as required.
8. Connect input signals to A INPUT and B INPUT connectors.
9. Press RESET push button .
10. Press RUN push button.
11. Allow processing to continue for a suitable length of time. The TIME CONSTANT indicator and lamp enable approximate measurement times to be calculated. (See Paragraph 3-40) .
12. Vertical calibration of the display is shown on the DISPLAY SENSITIVITY panel in V^2/CM .

3-49 Probability. Probability measurements give a statistical assessment of the amplitude characteristics of the signal under investigation. Measurements can be made of Probability Density Function (pdf) or the integral of the pdf, the Cumulative Distribution Function (cdf). Probability measurement procedure is as follows:

1. Set AC/DC switch for coupling required. When the input signal is dc or has frequency components less than approximately 1 Hz, the DC position must be selected.
2. Set A INPUT RMS VOLTS RANGE switch for level of signal being investigated (see Paragraph 3-8).
Horizontal calibration of the display is provided by the red scale (HORIZ V/CM PROBABILITY) on the A INPUT switch.
3. Set FUNCTION switch to PROBABILITY DENSITY (pdf measurement) or to PROBABILITY INTEGRAL (cdf measurement).
4. Set AVERAGING switch to SUMMATION or EXPONENTIAL as required. Generally, SUMMATION averaging will be found to be most useful. (With EXPONENTIAL averaging the vertical axis of the display is uncalibrated).
5. Set TIMESCALE switch for sampling rate for signal being measured: For TIMESCALE switch settings below $333\mu S$, sampling is limited at the fixed period of $333\mu S$.
In Probability measurements the TIMESCALE setting determines the sampling rate only and does not provide calibration of the horizontal axis of the display as in other modes
6. Set NUMBER OF SAMPLES N/TIME CONST MULTIPLIER N switch as required .
7. Connect input signal to A INPUT connector. (Probability measurements cannot be made on the B channel.)
8. Press RESET push-button.
9. Press RUN push-button.
10. Check that the display adequately fills the CRT screen horizontally. If not, adjust A INPUT HORIZ V/CM PROBABILITY switch for suitable display.
11. Allow processing to continue for a suitable length of time. The TIME CONSTANT indicator and lamp enable approximate measurement times to be calculated. (See Paragraph 3-40).
12. For vertical calibration of the display see Paragraph 3-14.



COMMUNICATION SYSTEMS

LABORATORY II

Noise Generation, Representation and Measurement

Introduction:

In this laboratory, you will examine two different noise generators to explore some of the properties of random noise waveforms and to learn something about the differences and similarities of the two different noise sources. These generators generate noise on two different principles—one amplifies the noise produced by a special noisy vacuum tube, while the other bases its noise on the seemingly scattered results of a digital filter with feedback.

Text References:

Chapter 5 in John G. Proakis and Masoud Saleh, *Fundamentals of Communication Systems*, Upper Saddle River, NJ: Prentice-Hall, 2005.

Chapter 6 in L.W. Couch II, *Digital and Analog Communication Systems, 6th ed.*, New Jersey, Prentice-Hall, 2001

Chapter 4 in *Communication Systems Engineering, 2nd ed.* by Simon Haykin

Chapters 4 and 5 in *Principles of Communications, 4th ed.* by R.E. Ziemer & W.H. Tranter.

Chapters 10 and 11 in *Modern Digital and Analog Communication Systems, 3rd ed.* by B.P. Lathi.

Preparation:

1. Read through carefully the material in the Appendix to this lab.
2. Find the noise-equivalent bandwidth of an n th order Butterworth filter for which

$$|H(f)|^2 = \frac{1}{1 + [f/f_{3dB}]^{2n}}$$

where f_{3dB} is the 3 dB frequency of the filter. What happens in the limit as n becomes arbitrarily large.

3. What is the period of a pseudorandom noise sequence (PRNS) of length $2^{15} - 1$ produced by the Wavetek 132 generator? What then is the spacing in the line spectrum of its output?

Apparatus:

- 1 - GR-1383 Random Noise Generator
- 1 - Wavetek 132 Pseudorandom Noise Generator
- 1 - True RMS Voltmeter
- 1 - Oscilloscope
- 1 - Spectrum analyzer
- 1 - Signal Generator
- 1 - Krohn-Hite 3202 filter unit




CAUTION**CAUTION****CAUTION**

Spectrum analyzers are very expensive, delicate and sensitive pieces of equipment which can be very easily abused. Make sure that at all times the signals you apply to the input does not exceed the maximum allowable input level noted on the front of the unit. If you are unsure of a signal level, measure it on your oscilloscope or with a voltmeter before you apply the signal to the spectrum analyzer.

Procedure:**Part I:** Physical Noise (The GR 1383)

1. Display the output of the generator on the oscilloscope. Observe the effects of varying the amplitude control. View the noise generator's output on the spectrum analyzer as well (don't forget the spectrum analyzer's maximum input levels!) Observe the effects of different settings of the video filter. Record and compare typical time and frequency domain waveforms. Observe (i) whether the obtained waveforms have a DC-component, (ii) the bandwidth of the output, (iii) whether the bandwidth can be changed, (iv) whether the oscilloscope can be synchronized to the noise signal, and (v) whether the noise signals have clearly defined amplitude levels.
2. Determine what the "output level" meter on the GR 1383 actually measures.
3. Is the noise produced by this generator "white" in any reasonable sense?

Part II: Simulated Noise (The Wavetek 132)

1. Set the generator to generate a pseudorandom signal. Observe the output both in time and frequency, varying the various control settings of "sequence length", "noise frequency", and  and . Record in a sketch the various waveforms obtain. Explain your observations.
2. Display the  waveform on the oscilloscope. Observe the effects of varying the DC offset control on the rear panel. Compare the waveform with a typical waveform obtained from the GR 1383. Comment on any significant differences. The "noise sync" output on the rear panel can be used to synchronize the scope. Why can the scope be synchronized when the Wavetek 132 is used but not when the GR 1383 is used. What effects does the sequence length have on synchronization? Can you synchronize to the noise signal when the oscilloscope is internally triggered?
3. Explain why is the *spectrum* of the waveform of part 2 is different than an "equivalent" waveform generated by a GR 1383.

Part Iii: Noise Power Measurements

1. Use a Krohn-Hite filter [which you may recall from your measurements in ELG 3175 is a fourth order ($n=4$) Butterworth filter when in MAX FLAT mode] connected as a bandpass filter, with a centre frequency of 35 kHz. Use the GR 1383 to provide an input to the filter. Observe the output waveform. Determine its midband gain and the 3 dB bandwidth of the filter.
2. Using a *true RMS voltmeter*, measure the output voltage. After correcting for gain, verify the proportionality between noise power and bandwidth. Are your results in consistence with the theoretical expected result? Explain.

APPENDIX

GR1383 Random Noise Generator (extracts from Operating Manual)

Section I Introduction

1-1 PURPOSE

The Type 1383 Random-Noise Generator (Figure 1-1) provides a high level of electrical noise at its output terminals. This type of signal is useful in testing video- and radio frequency systems for operation in the presence of noise, and for measurement of the noise figure of such systems. The noise signal can be used directly, or it can be used to modulate the output of a signal generator. It is useful in making crosstalk measurements and in determining the effective bandwidth of filters. The 20 Hz to 20 MHz bandwidth of the noise makes it useful for even wide-band video.

1-2 DESCRIPTION

The 1383 Random-Noise Generator consists of a thermionic diode noise source with all-semiconductor amplifiers and power supply. The noise output is useful

over the frequency range from 20 Hz to 20 MHz. The noise output amplitude is indicated by a meter on the front panel, and an output attenuator permits reduction of the output by a total of 80 dB in 10 dB steps from a maximum of 1 V rms, open circuit. The output impedance is 50 Ω.

1-3 CONTROLS, CONNECTORS AND INDICATORS

The controls, connectors and indicators on the front panel of the 1383 Random-Noise Generator is listed and described in Tables 1-1

The OUTPUT connector on the front panel is a GR874® coaxial connector. If desired, it may be removed from the front panel and mounted on the rear of the instrument. A snap-in hole cover (P/N 0480 2470) at the rear should be moved to the front if this change is made, to cover the hole in the front panel.

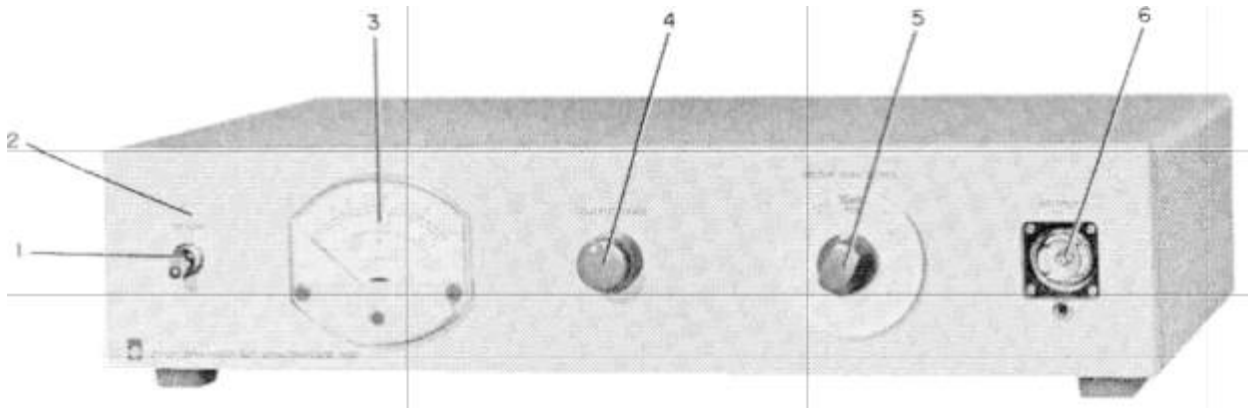


Figure 1-1 Type 1383 Random-Noise Generator

Table 1-1 Controls, Connectors, and Indicators on the Front Panel

| Figure 1-1 ref. | Name | Type | Positions | Function |
|-----------------|------------------|--|---|---|
| 1 | POWER | 2-position toggle switch | OFF, POWER | Energizes Instrument |
| 2 | — | Pilot lamp | — | Indicates when instrument is energized. |
| 3 | — | Meter | — | Indicates open-circuit output voltage. |
| 4 | OUTPUT LEVEL | Continuous rotary control | — | Varies output voltage. |
| 5 | METER FULL SCALE | 9-position rotary selector switch | 1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001 | Attenuates output in 10 dB steps. |
| 6 | OUTPUT 50 Ω | GR874® coaxial connector and ground jack | — | Connection to generator output. |

Section 4

Principles of Operation

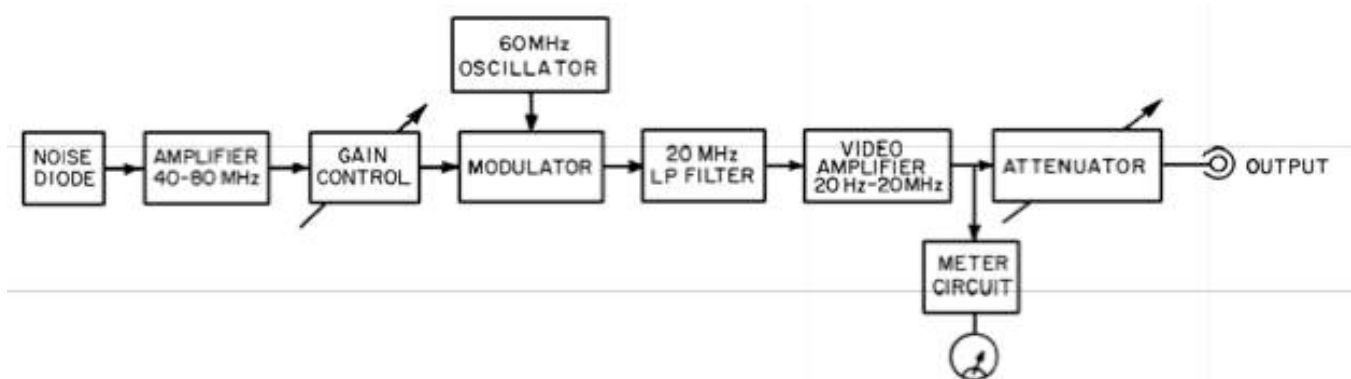


Figure 4-1. Block Diagram of the 1383 Random-Noise Generator

4-1 DESCRIPTION OF CIRCUIT

Figure 4-1 is a block diagram of the Type 1383 Random-Noise Generator. The noise source is a thermionic diode [a vacuum tube diode] operated in the temperature-limited mode. The noise output of such a diode operated in this way is exactly Gaussian, it is spectrally flat to extremely high frequencies, and its level is proportional to the square root of the DC plate current. To keep the amplitude stable, the filament current of the noise diode is controlled by feedback, to maintain a constant current. The noise output

of the diode is amplified in the frequency range from 40 to 80 MHz. By heterodyning against a 60 MHz local oscillator, the noise frequency is translated to the 0-to-20 MHz band. A sharp-cutoff low-pass filter ensures that higher noise frequencies and residual local-oscillator signals are removed. An AC-coupled amplifier is used to increase the power of that noise to the output level of 1 V. The voltmeter (which indicates the output level) and the output attenuator complete the circuit. All power supplies are regulated except the plate supply to the noise diode.

Section 5

Operating Procedure

5.1 GENERAL

With the instrument in position on the bench or installed in a relay rack, set the screwdriver-operated slide switch S502 (4, Figure 1-2) to the range corresponding to the power-line voltage. Connect the instrument to the line power, using the power cable supplied (P/N 4200-9622), and turn on the POWER switch.

Follow the instructions given in paragraphs 5.2 and 5.3 for setting the OUTPUT LEVEL and METER FULL SCALE

5.2 OUTPUT LEVEL CONTROL.

The OUTPUT LEVEL control is a continuous rotary control by means of which the output level can be set at any selected value between its maximum open-circuit value and zero. The open-circuit output voltage is indicated by the meter on the front panel, which must be read on the scale appropriate to the setting of the output attenuator (the METER FULL SCALE control).

5.3 METER FULL SCALE CONTROL.

The METER FULL SCALE control is the output attenuator and permits reduction of the output-voltage level by 80 dB from its maximum, in 10 dB steps. The voltage indicated is the open-circuit output voltage; when the output is loaded by 50 Ω , the output voltage is reduced by half. Decibel readings on the dial of the METER FULL SCALE control can be added to the decibel indication of the meter to obtain the open-circuit output voltage in decibels below 1 V.

5.4 OUTPUT CONNECTIONS.

The output connector on the front panel is a GR874, locking-type, coaxial connector. Generally, the output

should be taken by means of a mating GR874 connector into a closed coaxial system. Adaptors to other types of high-frequency connectors are available. The output connector can be moved to the rear panel if more convenient, as when the instrument is mounted in a relay rack (refer to paragraph 2.4 and Figure 8-2)

5.5 OUTPUT IMPEDANCE.

The output impedance of the 1383 is 50 ohms, $\pm 2\%$. The output can be short circuited without causing distortion of the output current. The maximum output current into a short circuit is 20 mA rms, with occasional peaks that may exceed four times that value.

5.6 MODIFYING THE OUTPUT

5.6.1 Producing Lower Levels.

GR874G fixed coaxial attenuators in reducing the output level beyond the range of the 1383 output attenuator (the METER FULL SCALE control). These units are available with attenuations of 3, 6, 10, 14, and 20 dB. They are designed for insertion in a 50-ohm line.

5.6.2 Generating Bands of Noise

It may be necessary to restrict the bandwidth of the noise output of the 1383. For inserting a tuned circuit or filter in series with the 50-ohm output, it may be convenient to use the GR874-X insertion unit, which permits totally-shielded connection of any circuit that will fit in its 2-inch long, 9/16 inch-diameter space.

5.6.3 Generating Higher Levels

The noise generated by the 1383 Random-Noise Generator can be amplified by any amplifier whose frequency range is adequate. The high crest factor of Gaussian noise must be

kept in mind in choosing the power rating of the amplifier so that the noise will not be clipped. In order that peaks of 3σ be passed without clipping, the amplifier must be capable of amplifying a sine wave without distortion to a power level 4.5 times greater than the average noise power desired.

5.6.4 Generating Noise at Higher Frequencies.

Noise at higher frequencies can be generated by using a double-balanced mixer to modulate a high-frequency carrier from an oscillator or signal generator. The result will be a band of noise 40 MHz wide, centered on the carrier frequency. (There will be a notch 40 Hz wide at the carrier frequency¹) The degree of discrimination against the carrier and the upper carrier frequency that can be used will depend upon the characteristics of the mixer used. Many such mixer units are commercially available.²

¹Perhaps wider, depending upon the characteristics of the mixer

²Relcom E. G., Mountain View, Cal.



Wavetek 132 VCG/ Noise Generator (extracts from Operating Manual)

SECTION 1

SPECIFICATIONS

VERSATILITY

Waveforms

Sine, square, triangle waveforms and analog noise  or digital noise .

Frequency Range of Signal

0.2 Hz to 2 MHz in 6 decade ranges

Ranges

| | |
|-------|------------------|
| ×10 | 0.2 Hz to 20 Hz |
| ×100 | 2 Hz to 200 Hz |
| ×1K | 20 Hz to 2 kHz |
| ×10K | 200 Hz to 20 kHz |
| ×100K | 2 kHz to 200 kHz |
| ×1M | 20 kHz to 2 MHz |

Function Outputs

Sine, square and triangle selectable, with 60 dB step attenuator in 10 dB steps and overlapping calibrated vernier, 50 Ω output impedance, 20 V peak-to-peak into open circuit and 10 V p-p into 50 Ω load from 50 Ω source impedance.

Sync Output

Greater than 1 V p-p square wave into open circuit at 600 Ω output impedance.

DC Offset

±5 V offset (±2.5 V offset into 50Ω load) controlled from rear panel; peak amplitude limited by the dynamic range of the amplifier output.

VCG—Voltage Controlled Generator

Frequency of the generator may be DC-programmed, or AC-modulated by external 0 to ±5 V signal. Voltage control circuitry is capable of 1000:1 deviation. The VCG amplifier has a 100 kHz bandwidth and a slew rate of 0.1 V/ms. The instantaneous frequency is the result of the sum of the dial setting and the externally applied voltage.

Stability

Short term +0.05% for 10 minutes
 Long term +0.25% for 24 hours

Percentages apply to amplitude, frequency, and DC offset.

HORIZONTAL PRECISION

Dial Accuracy

±2% of full scale, 1 Hz to 2 MHz

Frequency Vernier

One turn equals 1% of full scale.

Time Symmetry

±1% through ×100K range

VERTICAL PRECISION

Sine Wave Frequency Response

Amplitude change with frequency less than:
 0.1 dB from 0.2 Hz to 200 kHz

0.5 dB from 0.2 Hz to 2 MHz

PURITY

Sine Wave Distortion

Less than:

- 0.5% on ×10, ×100, ×1K, ×10K ranges
- 1.0% on ×100K range
- All harmonics 30 dB down on ×1 MHz range

Square Wave Rise and Fall Time

Less than 50 ns terminated into 50Ω

NOISE

Outputs

Pseudo-random analog or digital noise with a maximum of 20 V p-p excursion (open circuit) with 60 dB step attenuator in 10 dB steps and overlapping calibrated vernier.

Sequence Lengths

Push buttons on the front panel provide a sequence length of 2¹⁰-1, 2¹⁵-1 or 2²⁰-1

Noise Clock Frequency

Switch selectable noise frequencies are listed below.

| Clock Frequency | Analog Noise Bandwidth |
|-----------------|------------------------|
| 160 Hz | 10 Hz |
| 1.6 kHz | 100 Hz |
| 16 kHz | 1 kHz |
| 160 kHz | 10 kHz |
| 1.6 MHz | 100 kHz |

NOISE

FUNC Function Mode—Provides the selected waveform at the main output.

S/N Signal-to-Noise operation adds noise to a selected signal of constant amplitude. The signal-to-noise ratio is variable from 0 to +60 dB.

N/S Noise-to-Signal operation adds a selected signal to a constant amplitude noise. The noise-to-signal ratio is variable from 0 to +60 dB.

FM Frequency Modulation—Provides random modulation of the frequency of the generator. The S/N-N/S (dB) ratio control also controls the amount of frequency deviation.

NOTE

When noise is added to the signal output, specifications apply up to 200 kHz and the square wave rise time is derated by a factor of 10. In the clock range of 1.6 MHz, the maximum calibrated signal-to-noise ratio is 30 dB.

ENVIRONMENTAL

Temperature

All specifications listed, except stability, are for 25°C ±5°C. For operation from 0°C to 55°C, derate all specifications by factor of 2.

MECHANICAL

Dimensions

8.5 inches wide, 5.25 inches high, 11.5 inches deep

Weight

8 lbs net, 12 lbs shipping

Power

105 V to 125 V or 200 V to 250 V, 50 Hz to 400 Hz. Less than 15 watts.

NOTE

All specifications apply for frequencies obtained when dial is between 0.1 and 20 and at 10 V p-p into a 50 ohm load

It is possible to stop the generator from oscillating by applying a negative VCG voltage when the dial is already set at minimum frequency VCG inputs up to 30 V will not permanently damage the instrument

SECTION 2

OPERATION

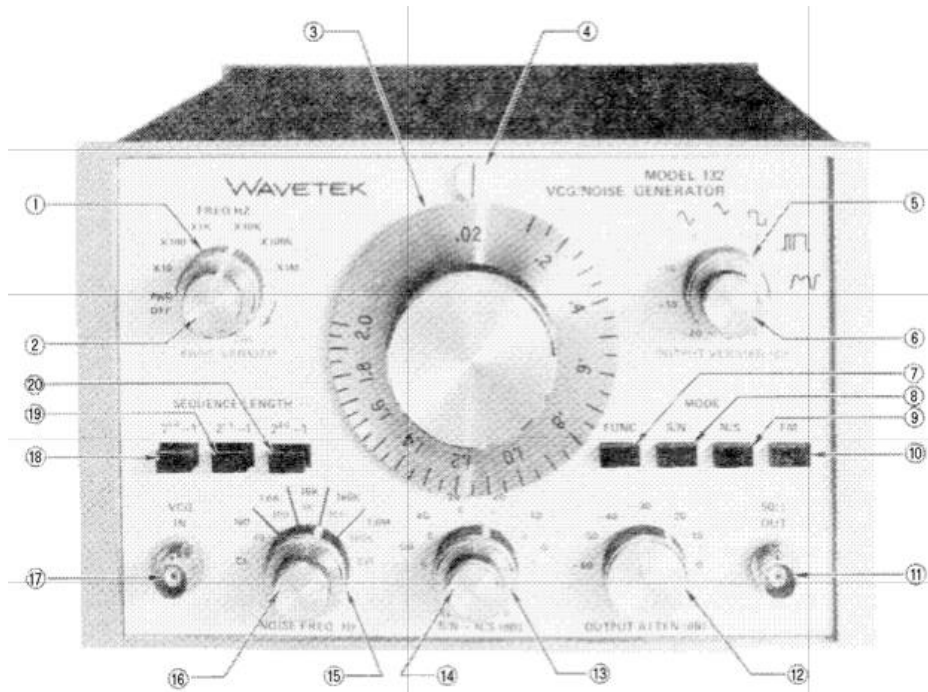




Figure 2-1 Operating Controls, Front panel

OPERATING CONTROLS

The operating controls and electrical connections for the Model 132 are shown in Figures 2-1 and 2-2. Each of the following paragraph numbers corresponds to a number appearing in Figure 2-1, front panel, or Figure 2-2, rear panel. The listing below discusses each control and its function.

FRONT PANEL

1. FREQ HZ/PWR OFF—Selects one of six decade ranges from ×10 to ×1M for generator frequency. This value multiplied by the frequency dial setting (3) gives the output frequency of the generator. Extreme counterclockwise rotation will place the switch in the PWR OFF position, turning off all power to the function and noise generators. This control has no affect on the noise frequency.

- 2. FREQ VERNIER**—Allows precision electronic control of the signal output frequency. A full turn of the control is approximately equal to 1% of full scale. When turned to the full clockwise position (CAL), settings on the main dial will be calibrated.
- 3. Frequency Dial** —Allows coarse control of the signal output frequency.
- 4. Frequency Index**—Indicates the frequency dial setting (3) by reading the dial position opposite the scribe line on the frequency index. The index is illuminated when power to the unit is on.
- 5. Function Selector**—Selects the desired function or noise output. To select sine, triangular, or square-wave waveforms, or  or  noise, the FUNC push button (7) must be depressed.
- 6. OUTPUT VERNIER (dB)**—Provides vernier control of 0 through -20 dB from the OUTPUT ATTN (dB)

setting (12). This is the fine adjustment for the output signal and will attenuate signal and noise.

MODE

- 7. **FUNC**—When depressed, this control allows the selected waveform or noise, as determined by the position of the function selector (5), to be present at the 50 Ω OUT connector (11). This push button must also be in the depressed position for the frequency modulation mode (10).
- 8. **S/N**—Depressing this push button allows a calibrated amount of analog noise to be added to the selected signal, either either sinusoid, triangular or square wave. The signal-to-noise ratio (S/N) is determined by the S/N — N/S (dB) attenuator control (13). When in this mode, the peak to peak signal amplitude is reduced internally, since adding noise to the signal would overdrive the output amplifier.
- 9. **N/S**—Depressing this push button allows a calibrated amount of the selected signal, either sinusoidal, triangular. or square wave, to be added to the analog noise. The noise-to-signal ratio (N/S) is determined by the S/N — N/S (dB) attenuator control (13). When in this mode, the peak to peak signal amplitude is reduced internally, since adding the signal to the noise would overdrive the output amplifier.
- 10. **FM**—Depressing this push button along with the FUNC push button (7) allows the selected signal, either sinusoidal, triangular. or square wave, to be pseudo-randomly frequency modulated, or jittered. The modulating signal is provided by pseudo-random analog noise, and the S/N_N/S (dB) controls frequency deviation. The bandwidth of the modulating signal is controlled by the NOISE FREQ Hz selector (15) and vernier (16).
- 11. **50 Ω OUT**—Provides the selected generator output function. The generator may operate into an open circuit providing 20 V peak to peak maximum, or into a 50 Ω load providing a 10 V peak to peak output.
- 12. **OUTPUT ATTEN (dB)**—Attenuates the output (both signal and noise, from 0 dB to -60 dB in six calibrated 10 dB steps according to the following table:

| Step Attenuator | Output peak to peak into 50Ω Load | |
|-----------------|-----------------------------------|------------------|
| Position | Maximum Vernier fully clockwise | Minimum Vernier* |
| 0 dB | 10 V | 1 V |
| -10 dB | 3 V | 0.3 V |
| -20 dB | 1 V | 0.1 V |
| -30 dB | 0.3 V | 0.03 V |
| -40 dB | 0.1 V | 0.01 V |
| -50 dB | 0.03 V | 0.003 V |
| -60 dB | 0.01 V | 0.001 V |

* The values in this table are approximate. The OUTPUT VERNIER (dB) (6) will reduce the output approximately 20 dB in all cases, as shown.

- 13. **S/N-N/S (dB)**—In the S/N mode, this control attenuates the analog noise from 0 to -50 dB in five calibrated 10 dB steps. The selectable signal amplitude remains constant, thus giving calibrated 0

to -50 dB signal-to-noise ratios. In the N/S mode, the signal is attenuated with the noise remaining unchanged, thus giving noise-to-signal ratios from 0 to -50 dB. The steps for this control are indicated in black numerals on the front panel.

- 14. **S/N-N/S (dB) Vernier**—Allows a calibrated fine adjustment of the S/N-N/S (dB) step attenuator (13) . This control is continuously variable over at least a 10 dB range. When added to the coarse control (13), this amount equals the total S/N or N/S ratio. Approximate values of attenuation are indicated in red numerals on the front panel.
- 15. **NOISE FREQ HZ**—This range control selects the clock frequency, or bandwidth for the digital, or analog noise, respectively. When using the digital noise function, clock frequencies from 160 Hz through 1.6 MHz (indicated in black numerals and letters on the front panel) are available. When using analog noise or the S/N, N/S modes, the bandwidth of the analog noise may be selected from 10 Hz to 100 kHz (indicated in red numerals and letters on the front panel). In the FM mode, this control establishes the bandwidth of the analog noise used for frequency modulation. There are four detent positions with an overlapping vernier control (16). With the vernier in the full clockwise position, the clock frequency, or bandwidth, is equal to the value printed to the right of the detent mark.
- 16. **NOISE FREQ HZ Vernier** As mentioned in number 15, this control provides a continuous, fine control between the detent positions of the coarse control. When in the full clockwise position, the clock frequency, or bandwidth, is equal to the value appearing at the right of the detent mark. As the knob is rotated counterclockwise, the clock frequency, or bandwidth, is decreased. In the full counterclockwise position, the actual value will be at least 10:1 (and as much as 100:1) lower than the value to the right of the detent mark.
- 17. **VCG IN**—This connector allows external voltage control of function generator frequency. Up to 1000:1 frequency change may be obtained. A positive voltage increases frequency and a negative voltage decreases frequency. Refer to "Operation as a Voltage Controlled Generator."

SEQUENCE LENGTH

- 18. **2¹⁰-1**—Depressing this push button will provide 1,023 counts of the selected clock frequency, or bandwidth, determined by the NOISE FREQ HZ controls (15 and 16), for generation of a digital, or analog noise pattern. At the end of each sequence, the pattern is automatically repeated.
- 19. **2¹⁵-1**—Depressing this push button will provide 32,767 counts of the selected clock frequency, or bandwidth, determined by the NOISE FREQ HZ controls (15 and 16), for generation of a digital, or analog noise pattern. At the end of each sequence, the pattern is automatically repeated.

20. **220-1**—Depressing this push button will provide 1,048,575 counts of the selected clock frequency, or bandwidth, determined by the NOISE FREQ HZ

controls (15 and 16), for generation of a digital, or analog noise pattern. At the end of each sequence, the pattern is automatically repeated.

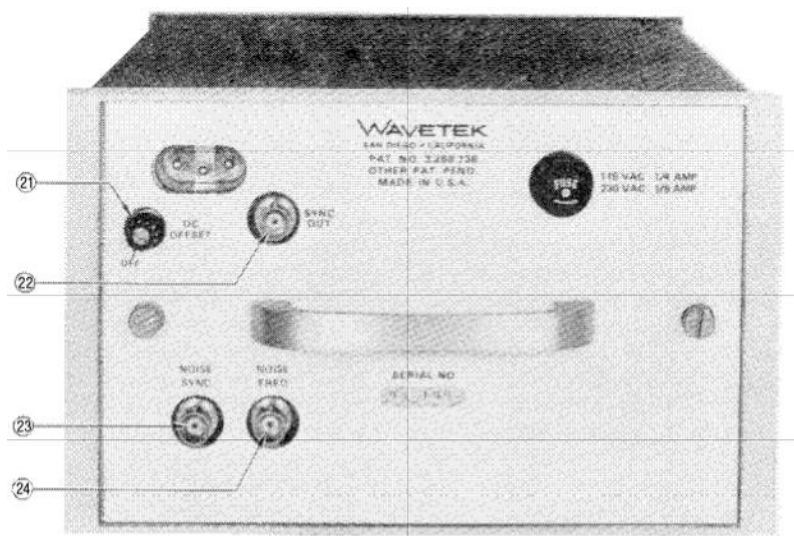


Figure 2-2 Operating Controls, Rear Panel

REAR PANEL

21. DC OFFSET—This control adjusts the ± 5 V base line above or below ground (± 2.5 V offset into 50Ω load). The OFF position gives normal vertical symmetry. Peak amplitude is limited by the dynamic range of the amplifier output.

22. FUNCTION SYNC—This connector provides a synchronizing signal output at the same frequency of the main generator; that is, at the same frequency as the sine, triangle or square wave. The amplitude is greater than 1 V peak to peak square wave into open circuit at 600Ω output impedance.

23. NOISE SYNC—This connector provides a synchronizing output signal for monitoring the digital or analog noise. A sync signal is generated at the beginning of each repetitive cycle for the selected sequence length of digital or analog noise.

24. NOISE CLOCK—This connector provides an external output of the basic clock frequency used to generate the digital sequence and analog noise.

OPERATION

No preparation for operation is required beyond completion of the initial installation previously stated in this section. It is recommended that a one-half hour warm-up period be allowed for the associated equipment to reach a stabilized operating temperature and for the Model 132 to attain stated accuracies.

Operation as a Calibrated Signal-to-Noise Source

1. Select the desired signal waveform by setting function selector to sinusoidal, triangular, or square wave.
2. Set frequency dial and FREQ HZ range multiplier for desired signal output frequency.

3. Depress MODE—S/N push button.
4. Select noise bandwidth by setting NOISE FREQ HZ control to desired range and fine adjusting bandwidth by turning the noise frequency Hz vernier control.
5. Select desired SEQUENCE LENGTH by depressing appropriate push button.
6. Select signal-to-noise ratio by setting S/N—N/S (dB) control to appropriate attenuation position and fine adjusting attenuation to desired value with S/N—N/S vernier control.
7. Select total output amplitude by setting OUTPUT ATTEN (dB) control to appropriate attenuation position and fine adjusting signal to desired amplitude with OUTPUT VERNIER (dB) control.

Operation as a Calibrated Noise-to-Signal Source

1. Select the desired signal waveform by setting function selector to sinusoidal, triangular, or square wave.
2. Set frequency dial and FREQ HZ range multiplier for desired signal output frequency.
3. Depress MODE—S/N push button.
4. Select noise bandwidth by setting NOISE FREQ HZ control to desired range and fine adjusting bandwidth by turning the noise frequency Hz vernier control.
5. Select desired SEQUENCE LENGTH by depressing appropriate push button.
6. Select noise-to-signal ratio by setting S/N—N/S (dB) control to appropriate attenuation position and fine adjusting attenuation to desired value with S/N—N/S vernier control.
7. Select total output amplitude by setting OUTPUT ATTEN (dB) control to appropriate attenuation

position and fine adjusting signal to desired amplitude with OUTPUT VERNIER (dB) control .

Operation as a Random FM Source

Before using the generator as a random FM source, please note the following.

The frequency of the generator is being varied or modulated by a changing voltage in the same way as described in "Operation as a Voltage Controlled Generator." However, instead of using a DC ramp, or AC signal, a random analog voltage is used. When the FM push button is depressed, the analog noise is injected internally into the VCG circuit; therefore, the modulation is created by random noise. The S/N–N/S (dB) knob controls the maximum amount of frequency deviation, since it controls the amplitude of the noise. Bandwidth of the FM signal is controlled by the NOISE FREQ HZ control. Using the generator in the FM mode may be accomplished as follows:

1. Select the desired signal waveform by setting function selector to sinusoidal, triangular or square-wave waveform.
2. Set frequency dial and FREQ HZ range multiplier for desired centre output frequency.
3. Depress MODE — FUNC and FM push buttons.
4. Select the bandwidth by setting NOISE FREQ HZ control to desired range and fine adjusting frequency by turning the noise frequency Hz vernier control.
5. Select desired SEQUENCE LENGTH by depressing appropriate push button.
6. Select signal frequency deviation by setting S/N – N/S (dB) control to appropriate attenuation position and fine adjusting attenuation to desired deviation with S/N–N/S vernier control.
7. Select output signal amplitude by setting OUTPUT ATTEN (dB) control to appropriate attenuation position and fine adjusting signal to desired amplitude with OUTPUT VERNIER (dB) control.

Operation as a Digital or Analog Noise Source

1. Set function selector to digital or analog noise position.
2. Depress MODE—FUNC push button.
3. Select clock frequency for digital or bandwidth for analog noise by setting NOISE FREQ HZ control to desired range and fine adjusting frequency by turning the noise frequency Hz vernier control.
4. Select desired SEQUENCE LENGTH by depressing appropriate push button.
5. Select noise amplitude by setting OUTPUT ATTEN (dB) control to appropriate attenuation position and fine adjusting noise to desired amplitude with OUTPUT VERNIER (dB) control.



ELG 4176

COMMUNICATION SYSTEMS

Fall 2016

Simulating a Communication System

Part 1: Introduction

Very often in practice, a proposed communication system is sufficiently complicated that its performance may not easily be found analytically. We may try to determine the performance in such cases by using performance bounds or estimates that can more easily be found, but these may not be sufficiently accurate for our needs. Since predicting a system's performance under anticipated operating conditions is a necessary part of the proper design of any system, this inability to determine a system's behaviour with acceptable accuracy cannot be tolerated. In such circumstances, an engineer usually resorts to performing a *Monte Carlo experiment* in the form of a computer simulation of the signals and components of the system to estimate how the real system would perform should it be built. In this computer laboratory problem, we will try to find the performance of a relatively simple system through such a simulation to illustrate this approach and to identify some of the factors to be concerned about in attempting to determine performance in this manner.

The computer simulation of any analogue signal processing system is fundamentally founded on the ability to represent any analogue signal through its samples taken at uniformly spaced instants in time at a sufficiently fast rate [as stated in the Sampling Theorem], and to represent the action of a subsystem on a continuous-time signal through some hopefully equivalent action on the samples of the signals. To be totally accurate, such a signal and system representation using signal samples is only possible exactly when the signals involved are all strictly bandlimited to some specific bandwidth or are otherwise suitably restricted in character. Unfortunately, the signals we must deal with in actual systems are seldom strictly bandlimited, with the result that we may only *approximately* represent a real communication system using discrete-time signals and systems. Done with appropriate care and understanding of the pitfalls, we can, however, sufficiently accurately approximate the behaviour of a communications system using appropriate sampling of the signals and random processes and appropriate processing of those sample. This approximation usually becomes increasingly accurate as we use larger and larger sampling rates on the signals. Larger sampling rates, however, require processing many more samples in the computer programs performing the simulation, which requires increased computing power or more time to complete the computations involved.

In this computer laboratory, we shall estimate through such a simulation the performance of a simple baseband binary digital communication system employing antipodal nonreturn-to-zero (NRZ) pulse amplitude modulation for the transmission of binary data over an additive white Gaussian noise channel, where the detector consists of passing the received signal through a first order lowpass filter (i.e., a *RC* lowpass filter), sampling the result at appropriate times and observing the sample's polarity. The system is shown in Fig. 1:

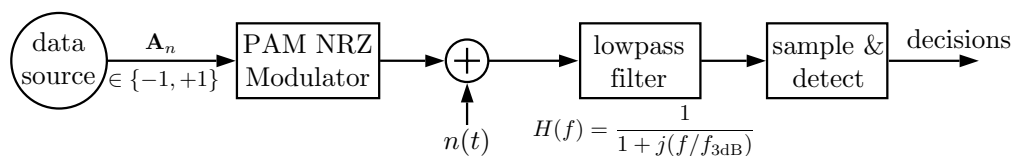


Fig. 1. The binary antipodal NRZ PAM transmission system to be simulated. The signal $n(t)$ is white Gaussian noise with a power spectral density $N_0/2$.

We shall attempt to determine the probability of a transmission error for this system versus the noise level. While this system is quite primitive by the standards of modern PAM systems, the issues we will have to deal with will be quite illustrative of the problems we can face in using simulation to determine the performance. To simulate this system, we begin by generating samples of the signal that would appear at the output of the modulator (samples of a randomly chosen realization of the signal since the actual signal depends of the data stream which is a random quantity), and then we model the actions of the blocks in the system by transforming the stream of signal samples to match as much as possible the actions of the blocks within the system. Due to the intersymbol interference produced by the lowpass filter, a precise expression we could evaluate for the probability of error even of this simple system cannot be given (although we can give very good approximations as we discuss a bit later). Throughout the discussion that follows, you should keep in mind that determining this performance is our ultimate aim: *we seek to produce a curve of the probability of bit error versus E_b/N_0 that would provide a very close approximation of the performance of the system if it was physically built and operated.*

Such a curve, based on the lab results, must be provided at the end of the report you will write. You will need to produce these curves to get a respectable grade on the lab. (See the discussion for Task 8 and Task 9 for more specific instructions.)

Part 2: Generating Random Numbers

Since many of the signals that appear in any communication system are random, in order to simulate the samples of such a process, we must be able to generate random values in a computer program. In particular, we find that we need to be able to generate a sequence of randomly chosen numbers, where the numbers are selected according to statistics of some known distribution. We begin by considering how this can be accomplished in a digital computer. Our aim in this discussion is to make you appreciate some of the limitations of such generators, including those that packages such as MATLAB employ, by creating your own such generator.

It must be recognized from the outset that any properly operating digital computer is a finite-state digital machine or automaton which has deterministic rules for all of its actions. To the extent that this is true, no digital computer can ever compute, or otherwise produce on its own a sequence of numbers that could be truly said to represent random sampling of the values of a random variable; to do so would require some ‘truly random’ input to the computer to influence the computation. Hence the very best one can hope for when we write a program to produce a sequence of so-called random numbers is to produce a sequence of numbers which *appears* (for the purposes to which these numbers are to be put) to be randomly chosen. We call such a sequence a *pseudo-random* sequence of numbers (and such sequences are used in many communication systems for other reasons, particularly in mobile communications). Several methods have been proposed and investigated for the production of sequences that appear sufficiently random for most applications, with significant new algorithms appearing in recent years. The issues involved in choosing a method are mainly ‘how random’ the generated numbers appear to be, and how computationally expensive it is to generate each number (since usually we must generate huge quantities of such random numbers). For many years, the most commonly used method was the so-called *multiplicative congruential method* which we will discuss and use here.[†]

The aim of basic methods for generating a pseudo-random number sequence including the multiplicative congruential method is to produce a ‘random’ choice of an integer in some range which is ideally the set of all possible integers the computer can naturally represent. For the multiplicative congruential

[†] Another more complex [but efficient to implement] algorithm to generate pseudo-random numbers was proposed by George Marsaglia several years ago and adopted in MATLAB. A short article discussing it is provided as Appendix B. Prior to MATLAB 5, the standard algorithm for generating random numbers was the multiplicative congruential method. With MATLAB 5, MATLAB switched to using Marsaglia’s algorithm, but then with MATLAB 7.4, the standard algorithm was changed again to yet another algorithm—the *Mersenne Twister* algorithm. This new algorithm will probably become the standard algorithm employed in most applications for generating pseudo-random numbers from the uniform distribution.

method, the range is the interval from 1 to some fixed number $M - 1$. In the method, we begin with some initial number x_0 in the 1 to $M - 1$ interval (the so-called ‘seed’), then multiply it by another number a and reduce the result modulo M .[‡] The result is the first pseudo-random number x_1 . This first pseudo-random number x_1 is then used as a new ‘seed’ for producing the next pseudo-random number x_2 : x_2 is x_1 multiplied by a and then reduced modulo M producing x_2 . If we let \widehat{m} denote the residue reduction of a quantity m modulo M , then repeatedly applying this procedure, we generate a sequence or pseudo-random numbers $\{x_1, x_2, x_3, \dots\}$, where $x_i = \widehat{ax_{i-1}}$. (In a modified version of this procedure, after we multiply by a we also added a constant c before the residue reduction. This modified version is termed the *mixed* multiplicative congruential scheme, where: $x_i = \widehat{[ax_{i-1} + c]}$.) Because in any arithmetic operation whose result will be reduced modulo M , we need not differentiate between two integers with the same residue reduction modulo M ,[§] we have the property that

$$x_i \equiv x_0 a^i \pmod{M},$$

where $a \equiv b \pmod{n}$ denotes the *congruency* modulo n relationship [from which is derived the name of this method; in the branch of mathematics known as number theory, two numbers are said to be congruent modulo n if they both have the same modulo n residue reduction]. Thus $x_i = \widehat{x_0 a^i}$ modulo M . As an example, if $x_0 = 1$, $a = 3$ and $M = 10$, the multiplicative congruential method would generate $\{1, 3, 9, 7, 1, 3, 9, 7, 1, 3, 9, 7, 1, 3, \dots\}$.

It has been found that proceeding in this simple way, we can generate numbers x_1, x_2, x_3, \dots which appear to the casual observer to be randomly selected. But in order for the resulting sequence to appear ‘sufficiently random,’ the parameters a and x_0 must be carefully selected to avoid two possible difficulties. To illustrate these difficulties, consider the following three selections of parameters and the resulting sequences:

- | | | |
|--------------------------------|---|--|
| (i) $x_0 = 1, a = 2, M = 8$ | → | $\{1, 2, 4, 0, 0, 0, \dots\}$ |
| (ii) $x_0 = 1, a = 2, M = 12$ | → | $\{1, 2, 4, 8, 4, 8, 4, \dots\}$ |
| (iii) $x_0 = 7, a = 2, M = 17$ | → | $\{7, 14, 11, 5, 10, 3, 6, 12, 7, 14, 11, \dots\}$ |

These three situations are examples of the three possible forms these sequences may take. The first, where the resulting sequence degenerates to zero is clearly to be avoided in a pseudo-random number generator. It occurs whenever the parameters allow zero to be generated, i.e., when for some i , $x_0 a^i$ is a multiple of M . The second and third cases, where following (perhaps) a few transient terms, the sequence repeats itself are the best sorts of sequence one can expect and thus include the desirable sequences. Obviously, if the period of this repetition was short, we would be hard pressed to see the sequence as random for almost any purpose. One objective, then, if we hope to be able to regard the sequence produced as a stand-in for a truly random sequence, is to select the parameters so the period of these repetitions is as large as possible.* The values for the parameters x_0 and a that yield the maximum period depend on the form of M . It turns out that if we insist on the longest possible period, then the resulting sequence of numbers will have only the form of the third example—that is to say,

[‡] To reduce a number p modulo M means to replace the number by another number from the integers $0, 1, 2, \dots, M - 1$ by repeatedly subtracting or adding M until a value in the range is obtained; this new number is termed “the residue reduction of p modulo M ”. Thus, for example, the residue reduction or 5 modulo 4, is 1, of -1 modulo 6 is 5, of 27 modulo 5 is 2, etc. Another loose way to describe the residue reduction is that it is the remainder one gets after dividing by M .

[§] For example, modulo 5, $2 \times l$ and $7 \times (l - 5)$ have the same residue reduction for any integer l , while $3 + l$ and $8 + (l + 10)$ have the same residue reduction.

* Any implementable algorithm for generating pseudo-random numbers has to be periodic due to the finite memory of any computer, and making the period as long as possible is one of the most basic goals in designing an algorithm. In the common implementations of the Mersenne Twister algorithm such as that used in the latest version of MATLAB (‘MT19937’ for 32 bit word length), the algorithm has a period of $2^{19937} - 1 \simeq 4.3 \times 10^{6001}$!

there will be no initial transient terms so that eventually x_0 recurs in the sequence at which point the sequence repeats itself. This result can be demonstrated using some standard results from the branch of mathematics known as *number theory*.

At this point, to understand the following material properly and undertake Task 1 and Task 2, it is very strongly recommended you go back to your earlier courses and review (in detail) how computers represent numbers, how fixed point computer arithmetic is handled, how we are able to deal with negative integer numbers, the difference between a signed and unsigned integer, etc. Without this knowledge, you cannot follow the next discussion nor undertake Task 1 and Task 2.

An important practical consideration in simulations is the computational cost of using the algorithm since simulations often require very many calls to the routines producing the numbers. In the multiplicative congruential method we must thus also be concerned with choosing parameters to make the computation of the modulo reduction of a number efficient to calculate. Now almost any digital computer which we could choose for a simulation uses what is in essence a binary representation for numbers and we can easily see that the residue reductions modulo a power or two of a number in binary form are particularly simple: the modulo 2^α reduction of a positive integer t is just the number corresponding to the least significant α bits in the binary representation of the number t . For example, the modulo reduction of $27=11011_2$ modulo $8 = 2^3$ has the binary representation consisting of the three least significant bits: $011_2 = 3$. If the computer stores its integers as α bit numbers in its arithmetic registers and memory, then residue reduction of the product ax_{i-1} needed in the implementation of the multiplicative congruential method is particularly easily and efficiently accomplished—just multiply a and x_{i-1} as two α bit integers and ignore the overflow that may develop in the multiplication process. If α was smaller than the bit length of integers on the computer, we would need to *mask* the results register to leave just the α least significant bits after doing the product (an extra step we should avoid for computational efficiency's sake). For these reasons, we will restrict ourselves to only the special case where $M = 2^\alpha$ (for α an integer greater than 2). It can be shown that the largest possible period for such a sequence when $M = 2^\alpha$ is always $2^{\alpha-2}$. In order to achieve this period, when $\alpha > 3$, the requirement is exactly that the following two conditions hold:

1. the initial seed x_0 and $M = 2^\alpha$ must be relatively prime (i.e., x_0 be odd, since all of M 's factors here are 2), and
2. that $a \equiv 3$ or 5 modulo 8.

Even if we require that we satisfy these two conditions, a great deal of latitude in the choice of the parameters remains. There are more considerations than simply the sequence period and computational efficiency that help narrow the choices. Another major factor influencing the choice of a arises from that fact that the numbers produced in succession by a random number generator should appear to be described by independent random variables and thus be uncorrelated. To obtain this apparent behaviour from the multiplicative congruential method no simple condition on x_0 , a and M can be given, but it has been found that a good 'rule or thumb' for choosing a is to satisfy the additional restriction:

3. select an a to be of the same order of magnitude** same \sqrt{M} .

Below is an example of a subroutine program in FORTRAN which was used to generate random numbers on a computer using 32 bit two's complement representation of integers such as the old IBM style mainframes (its essentially the program that was part of the standard set of subroutines provided with the FORTRAN compiler):***

** Two numbers are of the same order of magnitude if the scientific notation of the numbers involves the same exponent, or more simply, if the magnitude of their ratio is in the range of $\frac{1}{10}$ to 10.

*** The choice of parameters in this routine is now known to have some significant deficiencies statistically (e.g., the linear relationship $x_{k+2} \equiv 6x_{k+1} - 9x_k$ modulo 2^{31}) but it was adequate for many purposes in spite of the problems. This generator is now regarded by experts to be a 'horrible' generator with many flaws and to get good generators using the multiple congruential method, values for M that are not a power of 2 are needed—but that complicates the implementation.

```

SUBROUTINE PRAND (SEED)
C Program to generate pseudo-random numbers
  INTEGER SEED
  SEED = SEED * 65539
C Reduce results modulo  $2^{31}$ 
  IF (SEED) 5, 6, 6
  5 SEED = SEED + 2147483647 + 1
  6 RETURN
END

```

(For those unfamiliar with old style FORTRAN, the IF statement in the above simply represents the instruction to branch to statement labelled 5 if SEED is negative, and to branch to the statement labelled 6 if SEED is zero or positive.)

To understand how this routine works we must first recall how integer numbers are stored, and how stored bit patterns are interpreted. When we store a value as a pattern of α bits, we have 2^α possible values which we might naturally interpret as representing values 0 through to $2^\alpha - 1$ (as per an *unsigned* integer). Alternatively, to allow for positive and negative integers to be represented (as per a *signed* integer), we may use the two's complement form of representation where the values $0, 1, 2, \dots, 2^{\alpha-1} - 1$ are stored naturally (as a bit pattern where the most-significant-bit is 0), and values $-1, -2, -3, \dots, -(2^{\alpha-1})$ are stored as the two's complement of the corresponding positive number, corresponding to a bit pattern whose natural value is $2^\alpha - 1, 2^\alpha - 2, \dots, 2^\alpha - [2^\alpha - 2^{\alpha-1}]$ respectively (each of which has a bit pattern where the most-significant-bit is 1). In this way, arithmetic operations can be performed without regard to whether values are signed or unsigned. What value a stored bit pattern represents is merely a matter of interpretation—do we regard the stored value as a signed or an unsigned integer; the hardware that performs addition and multiplication does not know or care which interpretation will be taken. In FORTRAN, all values were integer variable treated as signed integers. The computers for which this program was written used 32 bit patterns to store all integers.

The subroutrine above is based on the desire to genrate a sequence which appears to be a succession of random selections of a positive integer from the set of those from 1 through $2^{31} - 1 = 2147483647$. It first simply multiplies the seed by 65539, ignoring the overflow and storing the least significant 32 bits of the result in the SEED variable (thereby computing the product modulo 2^{32}). The result is then reduced modulo 2^{31} (which corresponds to keeping the least significant 31 bits and zeroing the most significant bit) as follows: since any of these stored numbers with the most significant bit a 1 would be interpreted as a negative number, the most significant bit of the result is checked and if found to be a 1 (i.e., the computer believes SEED is negative), it is set to 0 by adding 2^{31} to it (it would be more efficient just to mask the bits, but FORTRAN had no mask instruction); since 2^{31} is not a legitimate integer on these computers (it is out of bounds), adding 2^{31} is accomplished by adding first $2^{31} - 1 = 2147483647$ and then adding 1. Note that it would NOT have been correct to get zero the most significant bit by replacing SEED with -SEED—this zeros the most significant bit but also changes the other bit which is not what we want to happen (that would not yield the product modulo 2^{31}).

Task 1. Examine the above subroutine and *in your own words*, provide a line by line explanation of how the subroutine implements the multiplicative congruential method, verifying whether the above explanation and comments in the program are indeed correct. What is the value of a and M in the subroutine's implementation of the method (explain how you know what these values are based on the code, not the comments in the code!)? Is the value of a implemented by this routine consistent with the above three requirements for generating desired maximum length sequences? Stated as simply as possible, what property would the initial value of the seed have to have for this subroutine to generate a maximal length sequence of seemingly independently selected numbers. What is the length of these longest possible sequences? If a computer generated numbers at the rate of 10^9 per second, how long would it take before repetition set in? What would happen if the initial seed chosen did not meet these requirements (be concrete and quantitative; carefully rewrite this code in your favorite language and experiment a bit with the choice of x_0 to see the effect)? In MATLAB, there is a corresponding

basic function `rand` to generate pseudo-random numbers. The period of the current version of `rand` is $2^{19937} - 1$. If numbers were produced using `rand` at the rate of 10^9 per second, how long (in years) would it take before repetition set in?

In your write-up of Task 1, you must clearly explain in your own words how the routine manages to perform the multiplication and modulo reduction required in the multiplicative congruential method. For example, in the context of this explanation, it would NOT be correct to state that in the fourth line above, “SEED is multiplied by 65539 and the result put back in SEED”; more than that happens, which is particularly significant for our purposes. Also, please note that the theory given above does NOT state that there are three conditions that must be satisfied for a maximal length sequence to be produced. Finally, if you think about the binary representations of the numbers when the initial seed is not even (but is an odd number multiplied by some power of 2), you should be able to determine theoretically the period when an initial seed violating condition 1 above is used (*Hint*: if the initial seed is even, then it is some odd number times the even number 2^β for some $\beta \geq 1$, and any product of an odd number a with the seed produces an odd number times 2^β , which is just the odd number in binary form shifted left by β places).

Task 2—Part A (The Code): When you have reviewed computer number representation and how computer arithmetic works, write your own *subroutine* similar to the one above to generate random numbers of a maximal length sequence that according to the above discussion would appear to be random. (Since we only discussed what happens for $M = 2^\alpha$, stick to such a choice of M .) Write the subroutine in a language such as C or Pascal (NOT in an interpreted language such as used in MATLAB). It should be a routine which has no parameters but returns a real number between 0 and 1. You should write the best possible generator for your programming system to first create uniformly distributed positive integers using the multiplicative congruential method, which are then scaled by dividing by the maximum possible integer [2^{31} for the above example [it is true that the maximum possible number for the Fortran program above is actually $2^{31} - 1$, but dividing by 2^{31} is a minor difference] so that it produces a sequence of random choices of a real number between 0 and 1—don’t just translate the above program verbatim or use any built-in random number generation routine. Think about how best to implement the multiplicative congruential method. Speed of execution is very important for a random number generator as the generator will be called many many times in a real simulation. This subroutine is to be the basis of the subsequent generation of random numbers—it is to be called whenever a random value between 0 and 1 is to be generated.

To make sure you don’t just copy the above parameters or those in Appendix B for the former MATLAB generator, you are *forbidden* to write a generator which uses an a of 65539 or 16807. You can experiment with different choices of parameters if you like, but in the write-up of this task, you must supply the program listing of a generator that is capable of generating at least several millions of numbers before repetition sets in. Because of the need for your generator to be used many tens of millions of times, you must give a good deal of attention to the efficiency of the implementation, so think how to minimize the number of machine language instructions that must be executed to generate a single random number. Don’t be afraid to examine the assembly language code that your compiler generates for your subroutine to see how the routine is translated to machine code.

Your write-up to this part of Task 2 should begin with an explanation of your consideration of how to efficiently implement a multiplicative congruential generator on *your* computer when the modulus is a power of two, and how then this influences your choice of the specific values for the parameters a and M . You must state exactly how you chose the values you picked and not merely give their values! Your thought process must be evident. You absolutely MUST then give the code for your subroutine in the Task 2 write-up (not merely as an appendix) and fully explain why it is a good pseudo-random sequence generator on your computer (with an appropriately chosen initial seed) and efficient in execution on your system. A program without a full explanation of how it was designed (especially how you sought to make the program execute as quickly as possible) is not acceptable, nor is code that is not in the form of a subroutine.

Task 2—Part B (Checking it Works): We would like to verify that a sequence so generated by your routine is effectively random with the desired uniform distribution. For this, we will simply examine the generator output for a significant number of successive values and test these numbers statistically in various ways against the model that the generator selects values randomly (with a uniform distribution) and independently each time. We know that all such generators produce a periodic sequence of numbers, and so to test the generator without bias, we should pick an arbitrary place in the sequence from which to start. To do this, we should start by picking an initial seed at random from those that could generate a maximal length sequence if used (e.g., if you wanted an odd 31 bit number as needed for the FORTRAN routine above, you could pick one at random by tossing a coin 30 times for the 30 highest order bits of the number [the lowest order bit must be 1]). Make such a random choice of an appropriate seed for your generator and then use your subroutine to generate 10,000 numbers starting from your seed, storing the results for analysis.

(a) Plot a histogram of the result using intervals 0 to .05, .05 to .10, etc. to see whether the numbers generated do appear to fit a uniform distribution on $[0, 1]$. Give the data for the histogram in table form as well. Find the average value of the numbers produced and see whether this is as one would expect if the numbers were drawn from a population that is uniformly distributed between 0 and 1 by viewing the histogram. (*Caution:* Do not simply use the MATLAB `hist` function applied to the data with a specification to use 20 bins [as in `hist(data,20)`] as this does NOT give the required histogram—this way, the `hist` function would NOT use the bins 0 to .05, .05 to .10, etc.)

(b) Simply staring at a histogram is not an objective way of confirming whether the frequency of different ranges of values is consistent with a particular distribution. More objective methods require employing statistical tests on the data. The usual statistical test that is applied to confirm the validity of a distribution as describing observed data is the so-called χ^2 -goodness-of-fit-test. It is discussed at length in virtually any statistics textbook (go back to your probability and statistics text and read up on the χ^2 -test; don't forget that in applying a χ^2 -test, you should choose bins so that at least 5 samples are in each bin to make the χ^2 -test statistic reasonably well approximated by a χ^2 random variable [see the example in Appendix A]). Apply the test to the histogram results you have above and determine whether, at a 5% level of significance, the results are consistent with the uniform distribution. Give a table for the calculation of the χ^2 -test statistic similar to that given in Appendix A. (Remember that at a 5% level of significance, there is a 5% chance that even if the distribution of the the numbers was uniform in the large, the test would conclude the observed results are not consistent with a uniform distribution hypothesis.) Comment.

(c) Determine how efficiently your routine generates these apparently uniformly distributed random numbers by timing how long it takes to make one hundred million calls of the subroutine to generate uniformly distributed random numbers. Time the generation twice: once where you do nothing with the numbers, and another time where you just compute their average (which requires generating the numbers and doing one addition per generated number). (To speed things along, it might help to tell your system to make the uniform numbers subroutine and 'in line' subroutine without any arguments [a global or static variable can hold the one parameter you should need], thereby avoiding the overhead normally associated with subroutine calls. The overhead of calling a subroutine is part of the time to generate the number, but may be reduced if the subroutine is made to be an 'in-line' subroutine [assuming your compiler has the option to produce in-line routines].) To make time results meaningful, you must document fully the system from which the values are derived, which includes specifying the hardware, the operating system, the compilers used to generate execution code, etc.—everything that affects how fast your code might execute on that computer. (*Caution:* Time just the actual generation of numbers and not the overhead needed to manage the generation. Take into account that in doing a loop where a subroutine is called once each loop, a program has to do things behind the scene besides calling the subroutine. The time to do these other things are not part of the time it takes to generate the random numbers. For example, if we write a loop that calls the subroutine we wrote once each loop pass and loops one million times, the time to execute this loop would be longer than a loop that calls the subroutine 10 times per loop pass and loops 100,000 times, yet both situations correspond to calling the subroutine 1,000,000 times. The difference is due to the different amount of time the loop management code gets executed. The time for loop management should be removed from the time spent executing

the subroutine call!)

(d) Apart from checking how uniformly distributed the values being generated seem to be, we would also like to verify that the numbers generated in sequence appear to be independently chosen each time. To do this would best be done by considering multidimensional histograms of the values produced to determine the joint distribution of successive values. This however is a bit unwieldy under the circumstances, so instead of doing this, we shall just check that the successive numbers generated appear to be uncorrelated by checking some of the sample covariances of the 10,000 generated numbers in the form of estimates of the autocovariance function.

For $m = 0, 1, 2, \dots, 100$, check to see whether the successive numbers r_1, r_2, r_3, \dots appear to be correlated by computing the *estimates* of the autocovariance function $C[m] \triangleq \mathcal{E}\{(\mathbf{r}_n - \mathcal{E}\{r_n\})(\mathbf{r}_{n+m} - \mathcal{E}\{r_{n+m}\})\}$ given by

$$\hat{C}[m] = \frac{1}{N - |m| - 1} \sum_{i=1}^{N-|m|} (r_i - \bar{x})(r_{i+m} - \bar{x}),$$

where N is the number of values selected (which is 10,000 here), and \bar{x} is the sample average given by

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N r_i.$$

We note that for a sequence of independent identically distributed random variables $\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3, \dots$ each uniformly distributed on $[0, 1]$, the autocovariance function is

$$C[m] = \begin{cases} \mathcal{E}\{(\mathbf{R}_i - \mathcal{E}\{\mathbf{R}_i\})^2\}, & \text{if } m = 0, \\ \mathcal{E}\{[\mathbf{R}_i - \mathcal{E}\{\mathbf{R}_i\}][\mathbf{R}_{i+m} - \mathcal{E}\{\mathbf{R}_{i+m}\}]\}, & \text{otherwise;} \end{cases} = \begin{cases} \frac{1}{12}, & \text{if } m = 0, \\ 0, & \text{otherwise.} \end{cases}$$

We also note that $\mathcal{E}\{\hat{C}[m]\} = C[m]$. Plot the computed estimates $\hat{C}[m]$ versus m (starting with $m = 0$, not $m = 1$). Is there any significant correlation or not?

Your write-up to this part of Task 2 should begin with an explanation of how you chose your initial seed and what value it was (consistent with the instructions), and then it should give the particular plots and tables from each subtask (including a table showing the calculation of any χ^2 -goodness-of-fit-test statistics) and the conclusions from each specific test individually about your implementation of a pseudo-random number generator. But then you must also consider the whole of Task 2, where *we have sought to write a pseudo-random number generator subroutine based on the multiplicative congruential algorithm that executes very quickly, seems to produce numbers according to the uniform distribution on $(0, 1)$, and produces a value with each call that is statistically independent of the results of any other call.* You have run some statistical tests on your generator to see whether your generator seems to function in this way. Your discussion of the task **must include a final overall evaluation as to whether you have succeeded**: what conclusion can you make overall about your generator subroutine from all the test results? Assuming your uniform number generator subroutine has been shown to be satisfactory, it must be used unchanged for the balance of this simulation. In the event that the tests lead you to the conclusion that your generator is not a good one, you must report that and then either do additional testing to verify further where your test results were merely statistical flukes, or you must make modifications to the generator and then all the testing above will have to be repeated to validate that the modified generator produces what appear to be independent and uniformly distributed selected numbers.

Now that we supposedly know how to generate on a digital computer seemingly uncorrelated pseudo-random numbers described by a uniform distribution (or rather we have programmed a generator and tested to some extent that it appears to produce a sequence of uncorrelated uniformly distributed random numbers on $(0, 1)$), we shall turn to the problem of how to generate what would appear to be numbers described by random variables with other sorts of distributions. We require this since the randomness in

a communication signal is usually not described by uniformly distributed random variables. For example, noise is most often modelled as having a zero mean Gaussian distribution so to simulate noise we would need to generate random numbers described by Gaussian random variables.

The usual way in which to accomplish this involves the use of numbers described by uniformly distributed random variables. As an example, if we wished to produce a random sequence of numbers $\{a_1, a_2, a_3, \dots\}$ corresponding to independent random variables whose values may be $+1$ with probability p and -1 with probability $q = 1 - p$, we could simply generate a sequence of numbers $\{r_1, r_2, r_3, \dots\}$ for the uniform $[0, 1]$ distribution and map each r_i to the $+1$ or -1 value according to whether the random number r_i was less than p or not respectively; such an approach could be used to describe the stream of data values output by some binary digital source. More generally, we can easily show that if \mathbf{X} is a random variable uniformly distributed on the interval $[0, 1]$ and $F_{\mathbf{Y}}(y)$ is the distribution function of some other random variable \mathbf{Y} whose inverse function exists and is $F_{\mathbf{Y}}^{-1}(y)$ (for $y \in (0, 1)$), then the distribution function of the random variable $F_{\mathbf{Y}}^{-1}(\mathbf{X})$ will be $F_{\mathbf{Y}}(\cdot)$. Thus, if we sought to generate a sequence of pseudo-random numbers supposedly described by random variables with a distribution function $F(x)$, we could do so by simply producing the sequence of numbers $\{F^{-1}(r_1), F^{-1}(r_2), F^{-1}(r_3), \dots\}$. (We could equally well replace the inverse of the distribution function by the inverse of the complementary distribution function and obtain the same effect.)

As an example of this, suppose we sought to generate a sequence of numbers described by independent identically distributed random variables with the basic Rayleigh distribution, i.e., governed by the density function

$$f_{\mathbf{X}}(x) = xe^{-x^2/2}u(x).$$

The corresponding distribution function is

$$F_{\mathbf{X}}(x) = (1 - e^{-x^2/2})u(x)$$

whose inverse function [defined only on $[0, 1]$] is

$$F^{-1}(x) = \sqrt{-2 \ln(1 - x)}.$$

If $\{\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \dots\}$ is a sequence of random variables uniformly distributed on $(0, 1)$, then $\{\sqrt{-2 \ln(1 - \mathbf{r}_1)}, \sqrt{-2 \ln(1 - \mathbf{r}_2)}, \sqrt{-2 \ln(1 - \mathbf{r}_3)}, \dots\}$ will be a sequence of numbers described by the Rayleigh density function above.

This approach can have its difficulties as it may be difficult to compute the inverse of the distribution function we would like to simulate. For example, if we sought to generate zero mean unit variance Gaussian random numbers, we would need to find the inverse function of $1 - Q(x)$ or of $Q(x)$, where

$$Q(x) \triangleq \int_x^{\infty} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$$

The inverse of such a function is difficult to obtain.^{††} Fortunately, simpler alternatives exist, one of which we shall now consider.^{‡‡}

The alternative we wish to use is based on the Central Limit Theorem and the knowledge that the mean of a random variable uniformly distributed on $(0, 1)$ is $\frac{1}{2}$, while its variance is $\frac{1}{12}$. The Central Limit Theorem predicts that if we were to add a number of independent random variables together, the distribution of the sum would approach a Gaussian distribution, and we observe that the distribution would very rapidly do so indeed in the central region. If we add together N independent random variables uniformly distributed on $(0, 1)$, the mean of the sum would be $N/2$ and the variance would be $N/12$

^{††} The inverse of the $Q(x)$ function can be approximated quite well numerically, but adequate such approximations are computationally expensive to evaluate and thus not used for this application.

^{‡‡} MATLAB uses another method to generate numbers described by the Gaussian distribution based on the *ziggurat algorithm*. It is better and more efficient than the method we will use.

(remember that the variance of a sum of independent random variables is the sum of the variances of those random variables), while the distribution would be close to being Gaussian. If we took $N = 12$, then the sum of the twelve random variables less 6 should for many purposes be an adequate simulation of zero mean unit variance Gaussian random numbers. (This approach fails to adequately simulate the distribution near its extreme possible values: the simulated Gaussian numbers here can range only between -6 and 6 while a truly zero mean unit variance Gaussian random variable has a probability of about 2×10^{-9} of being outside $[-6, 6]$.) Scaling the resulting (nearly) Gaussian numbers by σ and then adding μ will produce (nearly) Gaussian numbers with mean μ and variance σ^2 .

Task 3. Write a *subroutine* to generate a sequence of numbers that are described by independent zero mean unit variance Gaussian random variables using the above approach of adding twelve independent uniformly distributed random numbers (generated as per your tested program from Task 2). Computational efficiency is again important. The subroutine code and a discussion of the design considerations for the subroutine's implementation must be given as a central part in the write-up of this task. Use the subroutine to generate 20,000 Gaussian numbers using a *single* randomly selected appropriate initial seed integer, and plot a histogram of the results. Does the set seem to be well-described by a Gaussian distribution with zero mean and unit variance. What are the sample mean and variance? Are they consistent with the belief that we have a zero mean unit variance distributed set of numbers? Check with the χ^2 -test whether the obtained histogram is consistent with a zero-mean unit-variance Gaussian distribution hypothesis at the 5% level of significance. (Remember that in applying the χ^2 -test, the data should be separated into bins selected so that each bin contains at least five observed values. Also, to give the test some reasonable resolving power, use at least 10 to 15 subintervals (after any needed grouping) in determining the test statistic—more subintervals is better, fewer is not.) Give a table for the calculation of the χ^2 -test statistic similar to that given in Appendix A, and show how the expected values for the hypothetical distribution were determined so the reader can determine whether or not the test statistic was correctly computed.

Your write-up to Task 3 should begin with a presentation of the subroutine you have written to generate Gaussian pseudo-random numbers using the above described method, discussing your considerations for efficient execution. It then must fully present the test results (including a table showing exactly how the χ^2 -goodness-of-fit-test statistic was calculated (especially how the expected values in each selected bin was found). But then you must also consider the whole of Task 3, where that executes very quickly, seems to produce numbers according to the Gaussian distribution $N(0, 1)$, and produces a value with each call that is statistically independent of the results of any other call. You have run some statistical tests on your generator to see whether your generator seems to function in this way. Your discussion of the tasks must show how you attempted to accomplish all of the objectives and **it must include an overall evaluation as to whether you have succeeded.** Assuming your Gaussian number generator subroutine has been shown to be satisfactory, it must be used unchanged for the balance of this simulation; otherwise all the testing above will have to be repeated to validate that any modified generator produces what appear to be independent and $N(0, 1)$ selected numbers.

Part 3: Generating Random Signals

The above technique of generating independent Gaussian random variables can be used to generate a simulation of white Gaussian noise. Suppose we wished to run a simulation of a communication signal corrupted by additive white Gaussian noise with a power spectral density of $N_0/2$. One would probably proceed by generating samples of the signal component, and similarly simulate sampling the noise process. However there are a couple of difficulties that we face when we attempt to generate samples of a white noise process. Since the autocorrelation of such a process is $\frac{1}{2}N_0\delta(t)$ each random variable of this process is an independent zero-mean Gaussian random variable (they are uncorrelated and since they are all jointly Gaussian they are thus independent) with variance that of the signal's power. But the mean power of a white noise process is infinite, and how can we possibly generate infinite variance Gaussian random numbers? Secondly, we know from the Sampling Theorem that if we want to represent a signal with bandwidth B Hz through samples, the samples must be taken at a rate of at least $2B$ samples per second. The bandwidth of white noise is infinite, so this would seem to imply

we must generate samples at an infinite rate in order to represent the noise through its samples. Both of these problems demonstrate that we cannot possibly truly represent white noise through sampling.

Fortunately, we don't have to simulate all the components of white noise in the simulation of a communication signal. The part of the signal we must simulate can be found by considering the problem of representing the component of the communication signal other than the noise. These signals are much better behaved signals which although not strictly bandlimited, do have their power confined to some limited region of the spectrum. For example, for a binary NRZ PAM signal where the symbol values may be $+1$ or -1 with equal probability, the power spectrum of the transmitted signal

$$x_{\text{PAM}}(t) = \sum_n \mathbf{A}_n p(t - nT),$$

where

$$p(t) = \begin{cases} 1, & 0 \leq t < T; \\ 0, & \text{otherwise;} \end{cases}$$

is given by

$$S_x(f) = \frac{|P(f)|^2}{T} = T \text{sinc}^2 fT.$$

Thus for most purposes one can consider the signal to have negligible components beyond a few multiples of the symbol rate $1/T$. Thus to simulate this signal, we could simply generate samples of the signal taken at some rate N/T for N some value of about 6 to 10 or greater. For the N samples taken in the interval $[nT, (n+1)T)$, this is just N samples of \mathbf{A}_n , so we just need to 'randomly' pick a value for \mathbf{A}_n and then repeat it N times.

Assuming this is done, we see by sampling that we can describe any signal in the frequency range $-\frac{1}{2}N/T < f < \frac{1}{2}N/T$ (the value $\frac{1}{2}N/T$ is termed the *simulation bandwidth*). If this includes all the frequency components of the transmitted signal and any other frequency of importance in the system, we know that limiting the spectral components of the additive noise considered to just the components within this band will not affect the final system performance since the extraneous components would be removed ultimately. Thus rather than try to simulate a true white noise process through sampling—which the above discussion shows is really impossible—we should rather try to simulate a lowpass white noise process. Any bandwidth large enough to encompass all frequencies of any significance in the system should suffice. But therein lies another difficulty.

If a stationary Gaussian random process $\hat{\mathbf{n}}(t)$ is a white lowpass noise process with power spectrum $N_0/2$ and bandwidth B , any samples of the random process will be described by zero-mean Gaussian random variables with variance N_0B , which could be simulated as we have discussed. However, such a process has an autocorrelation/autocovariance function

$$R_{\hat{\mathbf{n}}}(\tau) = N_0B \text{sinc}(2B\tau).$$

and thus the successive samples of noise are likely correlated [$\hat{\mathbf{n}}(m\Delta T)$ and $\hat{\mathbf{n}}(n\Delta T)$ have a covariance of $R_{\hat{\mathbf{n}}}((m-n)\Delta T)$] and thus not governed by independent identically distributed random variables; the techniques discussed above, only applies to generating independent Gaussian random variables and thus could not easily handle directly generating a sequence of random numbers with the correlation required. But we may observe that if we use a sampling rate N/T and let $B = \frac{1}{2}N/T$, the covariance between $\hat{\mathbf{n}}(mT/N)$ and $\hat{\mathbf{n}}(nT/N)$ is $(N_0N/2T) \text{sinc}[2 \cdot \frac{1}{2}(N/T)(n-m)(T/N)] = (N_0N/2T) \text{sinc}(n-m)$ which is zero unless $n = m$. Thus for this bandwidth (which is the band represented by a sampling rate of N/T , the samples of the lowpass Gaussian white noise process *are* all described by independent zero-mean variance $N_0N/2T$ Gaussian random variables, which we may easily simulate with the techniques of generating pseudo-random numbers discussed above.

Task 4. Write code to generate a sequence of random numbers that corresponding to sampling a binary NRZ PAM signal, where the data symbols are described by independent identically distributed random variables which may equally likely be $+1$ or -1 . The code should generate N samples per symbol period, and cover M symbol periods, yielding in the end $K = NM$ samples. Use this code to generate

a possible sequence of signal samples and then plot the eye diagram of binary antipodal NRZ PAM on a simulation basis (one way to do this you may find easiest would be to just through the samples into a file in such a way that your favourite graphics program can read the file and make the needed plot; the Communications Systems Toolbox of MATLAB has a routine `commscope.eyediagram` (or the soon-to-be-obsolete `eyediagram`) to draw eye diagrams from such files, for example). A good eye diagram plot will usually draw more than one symbol interval (two would suffice). Make sure your eye diagram program deals correctly with the waveforms at the ends of the symbol intervals so that a full two or more symbol periods are drawn and the very last part of the eye in the final symbol period is not left out. Make sure all axis are properly labelled with the horizontal axis denoting continuous time (remember we are using this simulation to determine the behaviour of the continuous time system, and everything must be interpreted to that end—time is not measured as sample indices!). Plot the simulation approximated eye diagrams using at least two different choices of the sampling rate (expressed as samples per symbol period, N).

Remark: In this Task, we know what the samples of a signal are, and then to plot the signal, you most probably filled in between these samples by drawing a straight line between sample points. This is quite natural to do as it is easy to do, but we should recognize it NOT necessarily the most proper thing to do. Some problems we might identify with such a plot from samples may stem from this arbitrary means of interpolation between sample points. This is not the means of reconstructing a signal from samples that the Sampling Theorem provides. ■

Task 5. Write code to generate random numbers that could be used as samples of noise for a simulation of Gaussian white noise with a power spectral density of $N_0/2$ where samples are to be generated at a rate of N per symbol period. Write a program to use this code and the code from the above Task to generate the eye diagram for the sum of a bandlimited white Gaussian noise process and the NRZ PAM signal in the instances where the variance of the noise samples is 10% of the PAM signal's average power, and where it is 25%. What is the relationship between the variance of the noise samples and N_0 ; what values of N_0 does 10% and 25% of the PAM signal's average power correspond to. What is the value of E_b/N_0 in each case, and explain why this depends on N . ■

Part 4: Simulating Linear Time-Invariant Systems

The last aspect of the simulation of the system that we need to be concerned with is the simulation of the actions of a linear-time invariant system on signals (such as the first order lowpass filter in our system), given by the convolution integral

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t - \tau) d\tau.$$

The basis of such a simulation is largely the ability to approximate well an integral such as

$$I = \int_0^T f(x) dx \triangleq \lim_{N \rightarrow \infty} \sum_{n=0}^{N-1} \frac{T}{N} f(n[T/N])$$

with the finite Riemann sum

$$\sum_{n=0}^{N-1} \frac{T}{N} f(n[T/N])$$

when we select N sufficiently large.

Suppose we wish to determine $y(t_1)$. This is given by

$$y(t_1) = \int_{-\infty}^{\infty} h(\tau)x(t_1 - \tau) d\tau,$$

which we assume may be well approximated (by choosing N large enough as

$$\sum_{n=-\infty}^{\infty} \frac{T}{N} h(n[T/N])x(t_1 - n[T/N]).$$

If we assume the $t_1 = k[T/N]$, then we find that the approximation to $y(kT/N)$ becomes

$$\sum_{n=-\infty}^{\infty} \frac{T}{N} h(n[T/N]) x((k-n)[T/N]).$$

If we define $h_n \triangleq [T/N]h(n[T/N])$, $x_n \triangleq x(n[T/N])$ and $y_n \triangleq y(n[T/N])$, we see that this approximation becomes

$$y_k \simeq \sum_{n=-\infty}^{\infty} h_n x_{k-n}, \quad (*)$$

which is just the discrete-time convolution of two sequences. Simulation of the actions of a linear time invariant system often is accomplished by treating this equation as true with equality, and attempting to implement this relationship exactly or approximately. This view however can lead to some significant errors if we are not careful or don't make the appropriate interpretation of results based on it. We may be able to alter the values h_i a bit to have the sum of terms better approximate the continuous time convolution integral. In this we are concerned with the problem of how to build a discrete-time system to approximate the actions of a continuous-time system. This is a basic problem in the field of digital signal processing and has a number of different solutions you may learn about in ELG 4177/4577.

It should be noted that usually, the systems we seek to simulate are realizable systems and so must be causal, in which case we may simplify (*) by noting $h_n = 0$ when $n < 0$, leading to

$$y_k = \sum_{n=0}^{\infty} h_n x_{k-n}.$$

The fact that this is still an infinite sum can sometimes be bothersome, but often is not a difficulty. Take for example the system whose actions we wish to include in our system. This is a system with frequency response

$$H(f) = \frac{1}{1 + j(f/f_{3dB})}.$$

The impulse response that this corresponds to is

$$h(t) = \alpha e^{-\alpha t} u(t),$$

where $\alpha = 2\pi f_{3dB}$. The scaled samples of this impulse response taken at intervals T/N are given by

$$h_n = \begin{cases} \lambda e^{-n\lambda} & n \geq 0 \\ 0 & \text{otherwise} \end{cases} = \begin{cases} \lambda \beta^n & n \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\lambda = \alpha T/N$ and $\beta = e^{-\lambda}$. In terms of implementing a discrete-time LTI system with this infinite impulse response, we note that such a system can be implemented most simply in a recursive form by noting that

$$y_n = \beta y_{n-1} + \lambda x_n.$$

This is almost the simplest possible form of a discrete-time LTI system input-output relationship.

If we were to simply use this equation, we might note that if we applied the equivalent of $u(t)$ to the system, we would find an immediate reaction to the input: y_0 would be λ , when $y(0)$ would be zero. This is an artifact of how we chose to deal with the time-zero sample of the impulse response (where the impulse response has a discontinuity). To make the discrete system more like a real system would be expected to behave, we could incorporate a one sample delay in the previous system leading to an input output relationship of

$$y_{n+1} = \beta y_n + \lambda x_n.$$

We must remember that the discrete convolution equivalence has been arrived at by approximating an integral with the sum of the areas of rectangles. We expect that when the length of the rectangles

is sufficiently small, the approximation will be very good, but just how small must we make it, or equivalently, how fast we must sample the impulse response to get a good approximation. In the example we have here, we can get some idea of the effect of the sampling rate needs if we imagine what happens to a white noise process passing through the continuous-time system, and what happens to the simulated version of white noise passing through the discrete-time system.

If a Gaussian white noise process with power spectral density function $N_0/2$ is applied to a system with frequency response $H(f)$ as given above, the output will consist of a Gaussian random process with mean power

$$P_c = \frac{1}{2}N_0\pi f_{3\text{dB}}$$

and autocorrelation function

$$R_c(\tau) = \frac{1}{2}N_0\pi f_{3\text{dB}}e^{-\alpha|\tau|}.$$

Conversely, if a sequence of independent identically distributed random variables of variance $N_0N/2T$ is applied to a discrete-time system with impulse response $\lambda\beta^n u[n]$, the result is a Gaussian random sequence with average power

$$P_d = \lambda^2 N_0 N / [2T(1 - \beta^2)] = \frac{1}{2}N_0\pi f_{3\text{dB}} \cdot [\gamma / (1 - e^{-\gamma})] \quad \text{for } \gamma = 4\pi f_{3\text{dB}}T/N,$$

and autocorrelation function

$$R_d[n] = P_d \beta^{|n|} = P_d e^{-\alpha|n|T/N} = [\gamma / (1 - e^{-\gamma})] \cdot \frac{1}{2}N_0\pi f_{3\text{dB}} e^{-\alpha|n|T/N}.$$

This exactly matches the continuous-time result except for a scale factor: $R_d[n]$ is larger than $R_c(n[T/N])$ by the factor $[\gamma / (1 - e^{-\gamma})]$, which is close to 1 whenever γ is much less than 1, i.e., whenever N/T is an order of magnitude or more larger than $f_{3\text{dB}}$. Unfortunately, it may be too computationally expensive to use sufficiently large N to ignore this source of possible error.

The fact that this factor may be significantly more than unity is a potential cause for concern since it means that if the PAM signal's values after the filter in the simulation were to match the values of the continuous-time version, then for the same E_b/N_0 the error rate in the simulation would be larger than would actually occur in the real system we are trying to simulate. This would defeat the whole purpose of simulation which we must remember is to determine the performance of the physically real system. In fact, both signal and noise components may be scaled in the simulation relative to the real system and this must be considered and compensated for in some manner. For example, if for a given choice of simulation parameters, the signal levels of the simulated signal was twice its actual value (making E_b in the simulation four times larger than it would be in reality), and the value of the noise variance was double what it should be to match reality, then the 'simulation E_b/N_0 ' is 3 dB higher than it should be; this would mean that if we used the simulation to determine what E_b/N_0 it takes to obtain a given P_e , the simulation would indicate values which are 3 dB lower than it would actually require. The scaling of signal and noise would not be a problem if the simulated PAM signal's sample values at the output of the discrete time filter were larger by the same factor (making the simulation produce simply an amplified form of signal and noise in comparison to the continuous-time system we are attempting to model). Is this the case (or nearly so)? The following task should be completed with this issue in mind:

Task 6. Compute the factor $\gamma / (1 - e^{-\gamma})$ in the case where $f_{3\text{dB}} = 1/T$ and $N = 10$ (corresponding to a simulation with 10 samples/symbol-period). Work out the continuous-time response of the lowpass filter to $u(t)$, and compare this to the response of the 'equivalent' discrete-time filter to $u[n]$. Compare these results in the light of the comments in the above paragraph. Scale the responses so that the responses tend to the same value as $t \rightarrow \infty$ and $n \rightarrow \infty$ and plot these two curves on the same graph to see if their shapes accurately match. (*Caution:* Make sure that you are using matching time scales for for the discrete time and continuous time plots; remember there are N samples per symbol period T , not 1, so the discrete signal are samples every T/N seconds.) With such a scaling factor, would the computed eye diagram samples match what they should be in this simulation where the filter input is an NRZ signal? Would the same conclusion follow if the filter input was not an NRZ signal, but used, say, a triangular pulse? Is the discrete filter's response to simulated noise the same as the continuous time filter's response

to noise? Considering both signal and noise, is 10 samples/symbol period then adequate for an accurate simulation (in the NRZ case)? Based on these results, what accuracy (in dB) could be expected in the value of E_b/N_0 required to achieve a given P_e ? Can you predict what the minimum value N should have if you wish the accuracy to be better than 0.05 dB? [Answering this is a very important part of this laboratory, and is needed to understand the results you obtain in Part 5.]

Hint: Suppose the simulation of the filter was such that at the sampling instants, with an NRZ input, the signal component was exactly 110% of the signal the actual RC filter would produce, but the noise component was a Gaussian process whose mean power was 125% of the noise power in the actual RC filter's output. The higher signal levels in the simulation could be corrected by introducing an additional attenuation in the simulation filter of $1/1.1 = 0.90909$ (power gain $1/1.1^2 = 0.82645$), but that would change the noise power to $1.25/1.1^2 = 125/121 \simeq 1.03306$ times its correct value. The effect of this is to make the simulation appear to have more noise by $10 \log_{10}(125/121) \simeq 0.141$ dB than the actual system experiences. The performance curve for the simulation then would be the real world performance curve shifted by 0.141 dB. If we used the simulation to determine the value of E_b/N_0 required to obtain some certain P_e , we would produce a value that was 0.141 dB higher than what was actually needed—a 0.141 dB error or inaccuracy so-to-speak (assuming we can neglect the experimental error in measuring the simulation P_e discussed in Part 5). ■

Task 7. Write a program which will generate the samples of a NRZ PAM signal corrupted by noise as discussed above, and act upon these to simulate a first order lowpass filter according to our approximation. Pass the simulation of the NRZ PAM signal through the filter for $f_{3dB} = 1/T$ and plot the eye diagram of the output. Use $N = 10$ (i.e., 10 samples per symbol period). (To get the proper eye diagram, the samples out of the filter from the first several symbol periods should be discarded as they are not typical due to the transient effect of the beginning of the NRZ PAM signal). Compare the results with the eye diagram you should be able to predict for the continuous-time system. With the same noise levels added at the filter input as in Task 5, plot the eye diagrams again. Does the filter seem to have a beneficial effect on the signal? Plot the eye diagram at the filter output for two different choices of N (e.g., $N = 10$ and $N = 20$) with the variance of the input noise adjusted to produce an E_b/N_0 of 50 (17 dB) which in the case of $N = 10$ means a noise variance at the filter input equal to 10% of the signal power. ■

Part 5: Determining Performance

Once a simulation is available for all the system components and signals, we are able to use the simulation to determine many aspects of the system, such as the power spectrum of different signals, the dynamic range of signals, the probability of an error, etc.

In this simulation, we would simply like to determine the probability of a transmission error. To do this, we simply need to observe the filter output at the sampling instants we designate (which we take here to be the point where the eye diagram was found to be open the most), convert this to an estimate of the symbol value transmitted, and note how often in a simulation of the transmission of many bits we find the converted value differs from the symbol values to be sent. (Here we do this conversion simply by examining the polarity of the sample value.) This may require simulation of the transmission of thousands, millions, billions, or even more symbols. Obviously we do not want to waste a computer's time by sending huge numbers of symbols in a simulation, but we must send enough to be able to make reliable estimates of the error rate. How many are enough then? Clearly, if a communication system had an error rate of one in a million, we could not easily tell this by sending even 10,000 bits through the system (in a simulation or in real life) since they would all very likely be received without error—just as would happen if the error rate was 1 in 10^9 rather than 1 in 10^6 .

To see how many might be required for a reliable estimate, let us assume that the communication system is such that it is well-modelled as a binary symmetric channel (that is, the occurrence of an error for one transmitted bit does not depend on the value of the bit, nor is it related to the occurrence of an error in transmitting another bit). If L bits are sent through such a 'channel', the probability that l transmission errors occur is governed by a binomial distribution: the fraction of bits observed in error (defined to be the number of errors observed divided by the total number of bits transmitted) is a random variable with mean P_e (P_e is the actual probability of error) and variance $\sigma^2 = P_e(1 - P_e)/L$.

For a reliable estimate, the variance of the fraction should be a small fraction of P_e^2 , which requires that LP_e be significantly larger than 1. To be more concrete, we can determine the confidence interval of the estimate. Appealing to the central limit theorem, we can approximate the distribution of the fraction of bits in error as a Gaussian random variable with mean P_e and variance $P_e(1 - P_e)/L \simeq P_e/L$ (for $P_e \ll 1$). The 95% confidence interval of this random variable is from $P_e - 1.96\sigma$ to $P_e + 1.96\sigma$, so to obtain an accuracy no less than 20%, we would want

$$1.96\sigma \simeq \sqrt{P_e/L} < 0.2P_e.$$

This is true when $P_eL > 1.96^2/(0.2)^2 \simeq 100$. Since LP_e is the mean number of observed errors in total, we can formulate a rule of thumb for how long to observe errors: to obtain an estimate within about 20% of the actual value 95% of the time, you should set L so that at least 100 errors would occur. In practice, this means keep running blocks of symbols through the system until at least 100 errors have been found. For better accuracy (or to be more precise, for more reliable estimates), we should wait for even more errors to occur before terminating a simulation.

Task 8. Write a program to simulate the communications system to determine its error rate. Using 10 samples/symbol, $f_{3dB} = 1/T$ and various different noise levels, run simulations to allow you to plot the P_e versus E_b/N_0 for error rates of 10^{-1} to 10^{-5} (and 10^{-6} if you can manage it). Obtain at least five data points from which to determine a probability of error curve similar to Figure 8.26 in the text (the curve should not just be a piecewise linear connection of the five or more points at which the error rate was estimated but the expected sort of curve that you then fit to the available data points!). Run all simulations to obtain at least 100 errors. What is the 95% confidence interval we obtain for P_e based on observing 100 errors (assuming a binary symmetric channel model)? Give a table of the results for each data point showing the program settings for each point, the corresponding E_b/N_0 , and the 95% confidence interval to add error bars to your plotted data. Along with the plot of your data plot the performance of P_e [log scale] versus E_b/N_0 (in decibels) for the best possible demodulator of the received signal (i.e., the integrate and dump receiver) for which

$$P_e = Q(\sqrt{2E_b/N_0}).$$

From the simulation data, find an estimate of the value of E_b/N_0 for which the continuous-time system that we wanted to simulate would have $P_e = 10^{-4}$ and determine the penalty that using this suboptimum receiver has relative to the optimum receiver. Rerun the simulations for $N = 5$ and $N = 20$ to see if this changes the results significantly. Show the results from all three values of N on the same plot. From these results for different simulation bandwidths, how large would it appear that N has to be for the simulation results to accurately reflect the performance of the real system (i.e., is $N = 10$ sufficient, or perhaps $N = 5$ suffices)? To examine this question more thoroughly (without having to run many many long simulations), the analysis from Task 6 should help to predict how large N would have to be to keep the results to, say, within 0.05 dB of the true E_b/N_0 needed to obtain a given P_e by determining what values of N keep the computed signal and noise components at the filter output within 0.05 dB relatively to what they would be in the analogue system. Do the predictions from Task 6 match the results you get from your simulation runs?

We began this simulation project stating that we had to use simulation due to the complexity of the situation here where precise analytical analysis is too difficult. In fact for the system we have analyzed here, you have probably found that at the sampling instants, the PAM signal's value is very nearly 1 or -1 while the noise component is a zero-mean Gaussian random variable with variance $\frac{1}{2}N_0\pi f_{3dB} = \frac{1}{2}N_0\pi/T$. Since $E_b = T$, the probability of error of the system is easily found to be very well approximated^{†††} by

$$P_e \simeq Q(1/\sqrt{\frac{1}{2}N_0\pi/T}) = Q(\sqrt{2E_b/\pi N_0}).$$

^{†††} It is easy to show that in fact $Q(\sqrt{2E_b/\pi N_0}) < P_e < Q(\sqrt{1.992537E_b/\pi N_0})$.

The value 1.992537 is (approximately) twice the square of the one-sided eye opening at the filter output: $2(1 - e^{-2\pi})^2 \simeq 1.992537204$. The upper and lower bounds are obviously very close as they are separated by less than 0.01624 dB, a practically insignificant amount.

How close were your results to this? Make a single plot of performance you obtain from your simulation data (what you feel your data says about the actual system's performance), the above analytic approximate performance and the performance of the optimal integrate and dump receiver given above.

Your write-up to Task 8 must give the raw data of your simulation (the setting of the noise variance, the number of bits processed to encounter the 100 errors (or whatever value is chosen), the calculated E_b/N_0 , the estimated P_e with the confidence interval) for each data point; this is in addition to the plots. The final plot is the most important part of this whole simulation where you produce the performance curve determined from simulation and compare it to the calculated bounds and the ideal receiver performance (five smooth curves: one of the ideal receiver performance, one of the analytic performance approximation given immediately above, and then the three smooth curves through data points for each of the $N = 10$, $N = 5$ and $N = 20$ simulation results). Plotting smooth curves requires appropriate curve fitting of the data (check out MATLAB's nonlinear-curve-fitting tools). 25% of the lab grade is devoted to being able to generate a valid such plot.

If we reduce the filter's bandwidth to say $f_{3dB} = 0.215/T$ in an effort to reduce the noise corrupting the signal, it is much more difficult to analytically predict the system's performance, and simulation is needed to accurately predict the performance.

Task 9. Rerun the eye diagram generating code in Task 7 for the case of $f_{3dB} = 0.215/T$. Can you predict the system's performance from this easily? Rerun the simulations in Task 8 for $f_{3dB} = 0.215/T$ (one choice of N will be enough, but you can rerun the three choices from Task 8 if you have the time). Does this lower bandwidth improve the error rate? From the simulation data, find an estimate of the value of E_b/N_0 for which the continuous-time system that we wanted to simulate would have $P_e = 10^{-4}$ and determine the penalty that using this suboptimum receiver has relative to the optimum receiver and that of the case of $f_{3dB} = 1/T$. Make a single plot of performance you obtain from your simulation data (what you feel your data says about the actual system's performance), the results for the $f_{3dB} = 1/T$ case, and the performance of the optimal integrate and dump receiver given above.

Your write-up to Task 9 should be quite similar to the write-up of Task 8 in what is given. The final plot now has three smooth curves. 15% of the lab grade is devoted to being able to generate a valid such plot.

The Preliminary Report

By Oct. 21, each group must submit a preliminary version of the report on Tasks 1 to 3, to demonstrate they have a properly working version of the required routines, and thus have started significant work on the lab. Feedback on these submissions will be given, but there are no marks involved, though there may be a penalty if a brief report on this preliminary work is not submitted on time. The purpose is to ensure you have started the lab and understand the amount of effort that is needed to properly complete the lab.

The Final Report to be Submitted

Each group shall submit one joint report, with all the members of the group sharing the same mark (all members are responsible for each section of the lab, even though the work for different tasks may largely be done by particular members). This is based on all members contributing approximately equally to the work. The report cover must list the members of the group. Groups may *NOT* collaborate in completing tasks; discussion of problems encountered amongst the class members is encouraged, but after computations are attempted, not before. No collaboration between groups is allowed in writing the report. At the time of submission, all the program code for each part of the simulation must be sent via e-mail to Dr. Galko with a description of the system on which the code runs. The subject of the e-mail must be "Group XX Submission", where "XX" is replaced by your group number. Sufficient

information must be provided so that the programs can be rerun exactly as submitted to generate the data given in the report. The submission of the program code should take the form of a single ‘zip’ or ‘tar’ archive file containing the compressed version of each program. This archive must be named GroupXX.16.zip or GroupXX.16.tar, where XX is your group number (e.g., Group03.16.zip). In the archive, include subarchives/subdirectories of ALL the programs or spreadsheets associated with each task so the programs can be run to reproduce ALL your stated results precisely. The subdirectories must be labelled by task number. The precise version of code that generates the data must be supplied for each task for which code is written. This includes the code for generating plots (do NOT give long lists of the raw signal data that may be read by code that does plotting). Do NOT just indicate that code was modified in some way to produce the results, but give the modified program. Indicate what each file is for in some clear manner. Send the archive at the same time you submit your report. Your programs may be run to verify the results are correct.

In preparing the report for this computer laboratory, do *not* include a repetition of the discussion in these lab sheets or restate the Tasks. Rather, just report your results, including a detailed explanation of how they were obtained (so that it is clear that the task results were valid, giving the programs used where appropriate), the answers to the questions and and discussion on the significance and conclusions you can draw at each step (relevant to the task of predicting the real system’s performance from the simulation). When Tasks require that you write a program to achieve a result (as for example in Task 2), the program MUST be part of the answer to the task; programs that are not central to the issues raised in the tasks (e.g., a program you write to plot a graphical representation of an answer) should not be in the core of the report but MUST be given in an appendix. ALL PROGRAMS MUST BE PROPERLY COMMENTED, INCLUDING THE PROGRAMS JUST GENERATING PLOTS. Marks will be based on properly carrying out the tasks, the understanding of the issues demonstrated in the discussion, the quality of the presentation of the results (e.g., how well and properly graphical results were presented, etc.)—ultimately you are expected to produce a plot similar to Figure 8.26 in the text which gives the performances of the different system configurations as you estimate them from the simulation results; obviously the performance curve of a real system will be a smooth curve, so yours should be too. 40% of the grade for the report comes from validly generating these plots in Task 8 and Task 9! No marks are awarded for typing the report as opposed to neatly writing the report by hand (plots should be computer generated, not hand drawn). Grammatical, ‘typographical errors’ and similar mistakes however do count!

You may add more discussion of related matters than is demanded in the lab if you wish, but it has to be relevant to the aim of the laboratory. Be concise!

Finally, results must be reported honestly! There is a possibility in every statistical test that even when hypotheses are valid, a test for that validity will seem to require a conclusion that the hypothesis is invalid. Looking for a test to obtain the ‘correct’ outcome (or repeating a test over again with new data until the ‘proper’ conclusion is drawn) is invalid and nothing less than intellectual fraud (in this case, ‘cooking the numbers’ or ‘lying with statistics’). If you obtain results which seem aberrant *and you verify that you correctly conducted the test*, have the courage to report the results obtained and what conclusion you logically ought to draw. With a high probability, if there were 20 groups in the class, at the 95% level of significance, for any particular statistical test, about one group in the class should be expected to report a negative result when the hypothesis is valid. When everyone gets only ‘correct’ results, we have to conclude that it is likely that some have been dishonest. It has been very easy to discover this dishonesty in past years, and marks of zero will be the minimum penalty for such reports. With a smaller class size, it is easier to spot problems. Done correctly, the results of the simulation will match results that correct analysis will indicate, and many have been able to produce the correct results and analysis when due diligence was applied. Getting the wrong results has always been the result of programming errors beyond some basics such as assuming a value of N is fixed in calculating filter coefficients, using `hist` and other MATLAB routines incorrectly to generate histogram and other data, failing to correctly match discrete- and continuous-time, etc.

APPENDIX A

An Example of the χ^2 Goodness of Fit Test

In telephone system engineering, knowledge of the distribution of the load of telephone calls being placed is important when deciding how much capacity for handling calls is to be installed. It is believed that the number of telephone calls placed in a given time interval is accurately described by the Poisson distribution. In this description, the probability that k calls are placed in a given interval is

$$\frac{\mu^k}{k!} e^{-\mu},$$

where μ is a parameter of the distribution (which is the average number of telephone calls that could be expected in the interval). In an experiment, the number of calls placed in a two minute interval in a certain telephone office was monitored for 100 such nonoverlapping intervals with the results reported below. The telephone engineers would like to validate the Poisson model they use for the frequency of telephone calls, and so would like to test the above distribution against the observed data. No assumption is being made about the value of μ .

| No. of Calls Placed | Observed No. of 2 min. Intervals |
|------------------------|-------------------------------------|
| 0 | 1 |
| 1 | 5 |
| 2 | 16 |
| 3 | 17 |
| 4 | 26 |
| 5 | 11 |
| 6 | 9 |
| 7 | 9 |
| 8 | 2 |
| 9 | 1 |
| 10 | 2 |
| 11 | 1 |
| | 100 |

The parameter μ of the model is not known, so it must be estimated. Given the interpretation of μ as the mean, a good estimator of μ would be provided by the sample mean. From the data in the above table, the average number of calls per interval is found to be 4.20 which is our estimate of μ . With this choice of μ , we have then that the model predicts that the probability that there would be k calls initiated in an interval is given by

$$p_k = \frac{(4.2)^k}{k!} e^{-4.2}.$$

Thus for 100 intervals, we would expect $100p_k$ two minute intervals would be found in which k calls were placed on average (for $k = 0, 1, 2, \dots$). The table below lists the observed frequency and the predicted

frequency of k calls in a two minute interval.

| No. of Calls Made | Observed No. of 2 min. Intervals | Expected No. of 2 min. Intervals |
|----------------------|-------------------------------------|-------------------------------------|
| 0 | 1 | 1.5 |
| 1 | 5 | 6.3 |
| 2 | 16 | 13.2 |
| 3 | 17 | 18.5 |
| 4 | 26 | 19.4 |
| 5 | 11 | 16.3 |
| 6 | 9 | 11.4 |
| 7 | 9 | 6.9 |
| 8 | 2 | 3.6 |
| 9 | 1 | 1.7 |
| 10 | 2 | 0.7 |
| 11 | 1 | 0.3 |
| | 100 | 99.8 |

In order to use the χ^2 -goodness of fit test we cannot simply compute the χ^2 -statistic and compare it to a threshold determined from the distribution function tables for the χ^2 -distribution, since the χ^2 -statistic is closely described by the χ^2 -distribution only when at least 5 observations are expected in each cell and the cells account for all possible values the theoretical model predicts. To achieve this we simply group observations into cells (lumping the case of $k = 0$ and $k = 1$ together as well as the cases of $k = 8$, $k = 9$, $k = 10$, $k = 11$ and k -values for which there were no observed values (i.e., a group for $k \geq 8$; the expected number greater than 11 is just 0.1). [Which cells are grouped with which is somewhat arbitrary.] This produces the table below from which we see that the χ^2 -statistic for our grouping into 8 classes has the value 6.257.

| Observed No. of Samples, f_i | Expected No. of Samples, np_i | $(f_i - np_i)^2$ | $\frac{(f_i - np_i)^2}{np_i}$ |
|-----------------------------------|------------------------------------|------------------|-------------------------------|
| 6 | 7.8 | 3.24 | 0.415 |
| 16 | 13.2 | 7.84 | 0.594 |
| 17 | 18.5 | 2.25 | 0.122 |
| 26 | 19.4 | 43.56 | 2.245 |
| 11 | 16.3 | 28.09 | 1.723 |
| 9 | 11.4 | 5.76 | 0.505 |
| 9 | 6.9 | 4.41 | 0.639 |
| 6 | 6.3 | 0.09 | 0.014 |
| 100 | 99.9 | | 6.257 |

Since one of the parameters of the model was estimated, the χ^2 -statistic we have is supposedly well described by a χ^2 -distribution with $8 - 1 - 1 = 6$ degrees of freedom. If we test at the 5% level of significance, the threshold for the test is found from tables to be given by 12.59. Hence we find that we find that the observed data is consistent with a Poisson distribution hypothesis for the number of telephone calls placed in the chosen interval in the particular telephone office at the 5% level of significance. We have also estimated the appropriate parameter for the Poisson distribution to apply is $\mu = 4.2$.

Note that this test is NOT the same as having tested for the observation being described as a Poisson distribution with mean 4.20 (the value 4.20 is assumed fixed before the observations were made and is only exactly the sample mean by a lucky coincidence). If we were making this test, the computation of the χ^2 -statistic would be exactly as above, but this statistic would be described by a χ^2 -distribution with 7 degrees of freedom, for which the threshold increases to 14.07 at the 5% level of significance..

This test is also passed, from which we would be able to state that the observed data is consistent with a Poisson-distribution-with-mean-4.2 hypothesis for the number of telephone calls placed in the chosen interval in the particular telephone office at the 5% level of significance. (The difference here between the two tests is that we are testing the value of the mean as well as the distribution shape, while in the first test we are only testing for the shape of the distribution; had the sample mean not been 4.20, the computations of the χ^2 -statistic would be different for the two tests.)

P. Galko Winter 1992.
last modified Sept. 2016.

APPENDIX B

Extract from *MATLAB News & Notes*, Fall 1995 (with permission)
 ©1995 The MathWorks, Inc., All Rights Reserved.

Random thoughts

10⁴³⁵ years is a very long time

by Cleve Moler

Do you recognize this number?

0.21895918632809

If you're an avid MATLAB user, you've probably seen this number before, but don't remember it. It's the first number produced by the MATLAB random number generator with its default settings. Start up a fresh MATLAB session, set `format long`, type `rand`, and that's the number you get.

So, if all MATLAB users, all around the world, on all different computers, keep getting this same number, is it really "random"? No, it isn't. Computers are (in principle) deterministic machines and should not exhibit random behavior. If your computer doesn't access some external device, such as a gamma ray counter or a clock, then it must really be computing *pseudorandom* numbers. My favorite definition was given in 1951 by Berkeley Professor D. H. Lehmer, a pioneer in computing and, especially, computational number theory:

A random sequence is a vague notion... in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians...

Lehmer also invented the *multiplicative congruential* algorithm, which is the basis for many of the random number generators in use today. Lehmer's generators involve three integer parameters, a , c , and m , and an initial value, x_1 , called the *seed*. A sequence of integers is defined by

$$x_{k+1} = (a x_k + c) \bmod m$$

(The operation "mod m " means take the remainder after division by m .) For example, with $a = 13$, $c = 0$, $m = 31$, and $x_1 = 1$, the sequence begins with

1 13 14 27 10 6 16 22 7 29 5 3

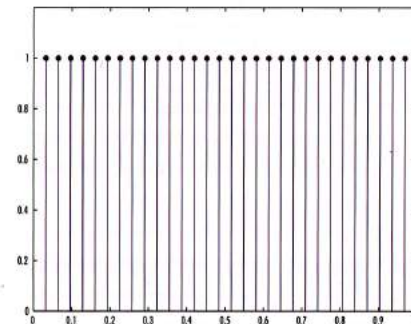
What's the next value? Well, it looks pretty unpredictable, but you've been initiated. So you can compute $13 \cdot 3 \bmod 31$, which is 8. The first 30 terms in the sequence are a permutation of the integers from 1 to 30 and then the sequence repeats itself. It has a *period* equal to $m-1$.

A pseudorandom integer sequence with values between 0 and m can be scaled by dividing by m to give floating-point

numbers uniformly distributed in the interval $[0, 1]$. Our simple example begins with

0.0323 0.4194 0.4516 0.8710 0.3226 0.1935 0.5161...

The histogram of this sequence is:



There are only a finite number of values—30 in this case. The smallest value is $1/31$; the largest is $30/31$. Each number is equally probable in a long run of the sequence.

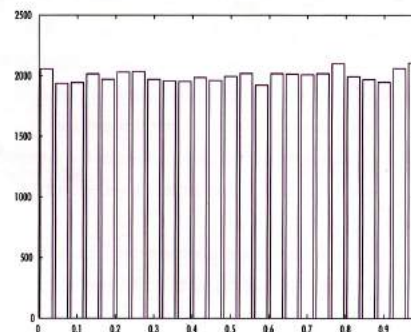
The uniform random number function, `rand`, in the current release of MATLAB, has similar behavior. It is a multiplicative congruential generator with parameters

$$a = 7^5 = 16807$$

$$c = 0$$

$$m = 2^{31} - 1 = 2147483647$$

These values were recommended in a 1988 paper by S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find" (*Comm ACM*, vol. 32). Here is a histogram of 50,000 values produced by `rand`.



"Nothing in nature is random... A thing appears random only through the incompleteness of our knowledge."—Spinoza

Each of the 25 bins contains roughly 2,000 numbers. We would see a similar picture for any other reasonable number of bins. Our generator satisfactorily passes this histogram test, which is the admission exam for uniform generators.

Like our toy generator, `rand` generates all real numbers of the form k/m for $k = 1 \dots m-1$. The smallest and largest are 0.00000000046566 and 0.99999999953434. It repeats itself after generating $m-1$ values, which is a little over two billion numbers. A few years ago that was regarded as plenty. It probably still is today, but it's getting a little skimpy. On a 75 MHz Pentium laptop, we can exhaust the period in fewer than four hours. Of course, to do anything useful with two billion numbers takes more time, but we would still like to have a longer period.

We've developed a replacement for our current `rand`. It will be part of MATLAB version 5. The new algorithm is based on advice from George Marsaglia, a professor at Florida State University, and author of the classic analysis of random number generators, "Random numbers fall mainly in the planes," (*Proc. Nat. Acad. Sci.*, vol 61, 1968).

Marsaglia's new generator does not use Lehmer's multiplicative congruential scheme. In fact, there are no multiplications or divisions at all. It is specifically designed to produce floating-point values. The results are not just scaled integers. And, it is *fast*. We get close to a "megarand"—a million random numbers per second—on our laptop.

In place of a single seed, the new generator has 35 words of internal memory or *state*. Thirty two of these words form a cache of floating-point numbers, z , between 0 and 1. The remaining three words contain an integer index i , which varies between 1 and 32, a single random integer j , and a "borrow" flag b . This entire state vector is built up a bit at a time during an initialization phase. Different initial states can be triggered by specifying different values of j .

The generation of the i -th floating-point number in the sequence involves a "subtract with borrow" step, where one number in the cache is replaced by the difference of two others.

$$z_i = z_{i+20} - z_{i+5} - b$$

The three indices, i , $i+20$, and $i+5$ are all interpreted mod 32 (by using just their last five bits). The quantity b is left over from the previous step; it is either zero or a small positive value. If the computed z_i is positive, b is set to zero for the next step. But if the computed z_i would be negative, it is made positive by adding 1.0 before it is saved and b is set to 2^{-53} for the next step. The quantity 2^{-53} , which is half of MATLAB's built-in constant `eps`, is called one *ulp* because it is one unit in the last place for floating-point numbers slightly less than 1.

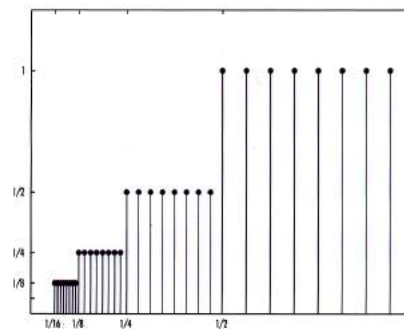
By itself, this generator would be almost completely satisfactory. Marsaglia has shown that it has a huge period—almost 2^{1430} values would be generated before it would repeat itself. But it has one slight defect. All the numbers are the result

of floating-point additions and subtractions of numbers in the initial cache, so they are all integer multiples of 2^{-53} .

Consequently, many of the floating-point numbers in $[0,1]$ are not represented.

The floating-point numbers between $1/2$ and 1 are equally spaced with a spacing of one *ulp*, and our subtract-with-borrow generator will eventually generate all of them. But numbers less than $1/2$ are more closely spaced and the generator would miss most of them. It would generate only half of the possible numbers in the interval $[1/4, 1/2]$, only a quarter of the numbers in $[1/8, 1/4]$, and so on. This is where the quantity j in the state vector comes in. It is the result of a separate, independent, random number generator based on bitwise logical operations. The floating-point fraction of each z_i is xored with j to produce the result returned by the generator. This breaks up the even spacing of the numbers less than $1/2$. It is now theoretically possible to generate *all* of the floating-point numbers between 2^{-53} and $1-2^{-53}$. We're not sure whether all of them are actually generated, but we don't know of any that can't be.

This graph illustrates what we're trying to accomplish, with one *ulp* equal to 2^{-4} instead of 2^{-53} .



The graph depicts the relative frequency of each of the floating-point numbers. A total of 32 floating-point numbers are shown. Eight of them are between $1/2$ and 1 and they are all equally likely to occur. There are also eight numbers between $1/4$ and $1/2$, but since this interval is only half as wide, each of them should occur only half as often. As we move to the left, each subinterval is half as wide as the previous one, but it still contains the same number of floating-point numbers, so their relative frequencies must be cut in half. Imagine this picture with 2^{53} numbers in each of 2^{52} successively smaller intervals and you will see what our new random number generator is doing.

With this additional bit fiddling, the period becomes something like 2^{1492} . Maybe we should call it the Christopher Columbus generator. In any case, it will be a *very* long time before it repeats itself. At one million per second, it will take more than 10^{435} years. ■

Cleve Moler is Chairman and co-founder of The MathWorks. His e-mail address is moler@mathworks.com.