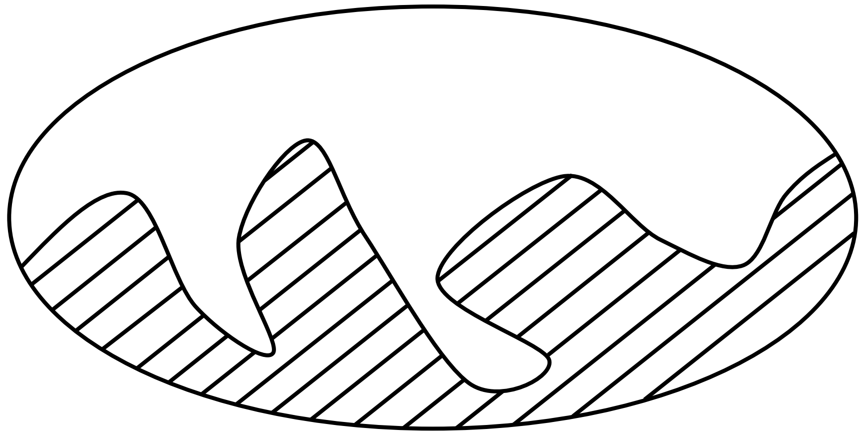
Simulation and Generation of Random Numbers

Principle of Monte Carlo Simulation

- Simulation techniques using RNGs bear the name of the city of Monte Carlo, the home of the world-famous casino
- An example demonstrating the power of MC simulation is to find the area of the following awkward region which may not permit an analytical calculation but can easily be solved by simulation



What’s Next?

- Generation of Random Numbers
- Generation of Random Uncorrelated/Corrected Sequences
- Pre-LAB 1 (BER Simulation of a Simple BPSK System)

Random Number Generation

- The key to implementing a MC simulation is the generation of sequences of random numbers which represent the sampled values of the input random processes

- The overall accuracy of the simulation results will depend on how good these algorithms are in faithfully reproducing the statistical properties of the random processes they are supposed to mimic

Generation of Uniform Random Numbers

- One such method to efficiently generate uniform random numbers, called the congruential or the power residue method, uses

$$\boxed{X(k) = [aX(k-1)+c] \mod M}$$

a is the multiplier such that 0<a<M

X(0) is the starting value or the seed and 0<X(0)<M

c is the increment, usually =1 or 0

M (>0) is the modulus and a large integer value

- The seed X(0) is provided by the user and is the only thing random

- It produces a sequence of integer values in $0 \leq X(k) \leq M-1$ which can be converted into a random sequence of uniform random numbers in [0,1] by

$$\boxed{U(k) = \text{Float}\left[\frac{X(k)}{M}\right]}$$

Generation of Uniform Random Numbers

- The most commonly used uniform RNGs are:

$$\boxed{\begin{array}{r} X(k) = 16,807X(k-1) \mod 2^{31}-1 \\ \text{for 32-bit machines with period} = 2^{31}-2 \end{array}}$$

$$\boxed{\begin{array}{r} X(k) = [69,069X(k-1)+1] \mod 2^{32} \\ \text{for 32-bit machines with period} = 2^{32} \end{array}}$$

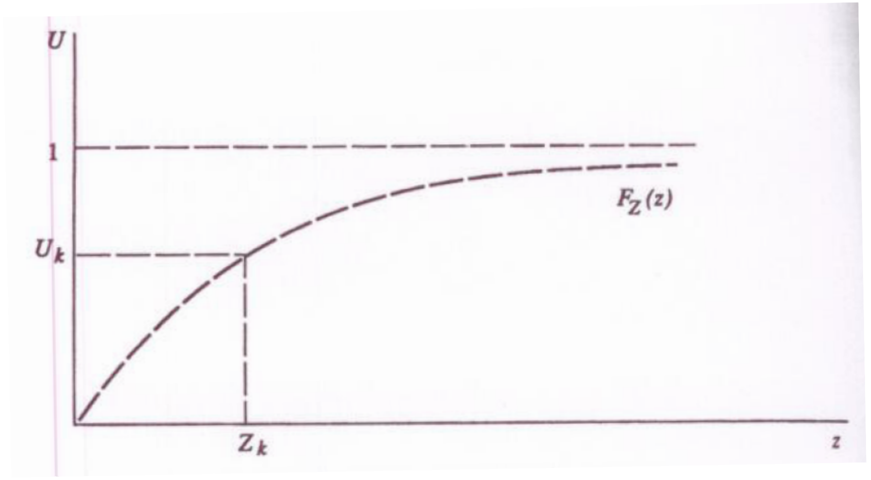
Generating Random Numbers from a PDF

- By applying a simple transformation to a uniform random variable U, we can generate a random variable Z with an arbitrary pdf $f_z(z)$

- Consider the transformation

$$Y = F_z(Z) = \text{the CDF of } Z$$

- It can be shown that Y has a uniform distribution in the interval [0,1]



Generating Random Numbers from a PDF

- We can thus generate Z using $F_z^{-1}(Y)$ where Y has a uniform PDF:

1. Generate U uniformly distributed in [0,1]

2. Output $Z=F_z^{-1}(U)$; Z has the pdf $f_z(z)$

Generating Gaussian Random Variables

- The simplest method of generating random numbers with a standard Gaussian PDF (μ_Y=0 and σ_Y^2=1) is via the use of the central limit theorem (CLT) according to the formula

$$\boxed{Y = \sum_{k=1}^{12} U(k) - 6}$$

- Since U(k) has a mean of 0.5 and variance of 1/12, it is easy to verify that Y yields a mean of 0 and variance of 1
- Whereas a Gaussian random variable has values ranging from $-\infty$ to ∞, the above formula produces values of Y in the interval [-6,6]
- More uniform numbers can be added to improve accuracy!

Generating Independent Random Sequences

- **Random Binary Sequences** – A random binary sequence $\{b_k\}$, where $b_k=0$ or 1, can be generated from a uniform sequence $\{U_k\}$ by

$$\boxed{b_k = \begin{cases} 1 & \text{if } U_k > p_1 \\ 0 & \text{if } U_k \le p_0 \end{cases}}$$

where $p_0=P[b_k=0]$

- Sampled values of a random binary waveform can be generated by

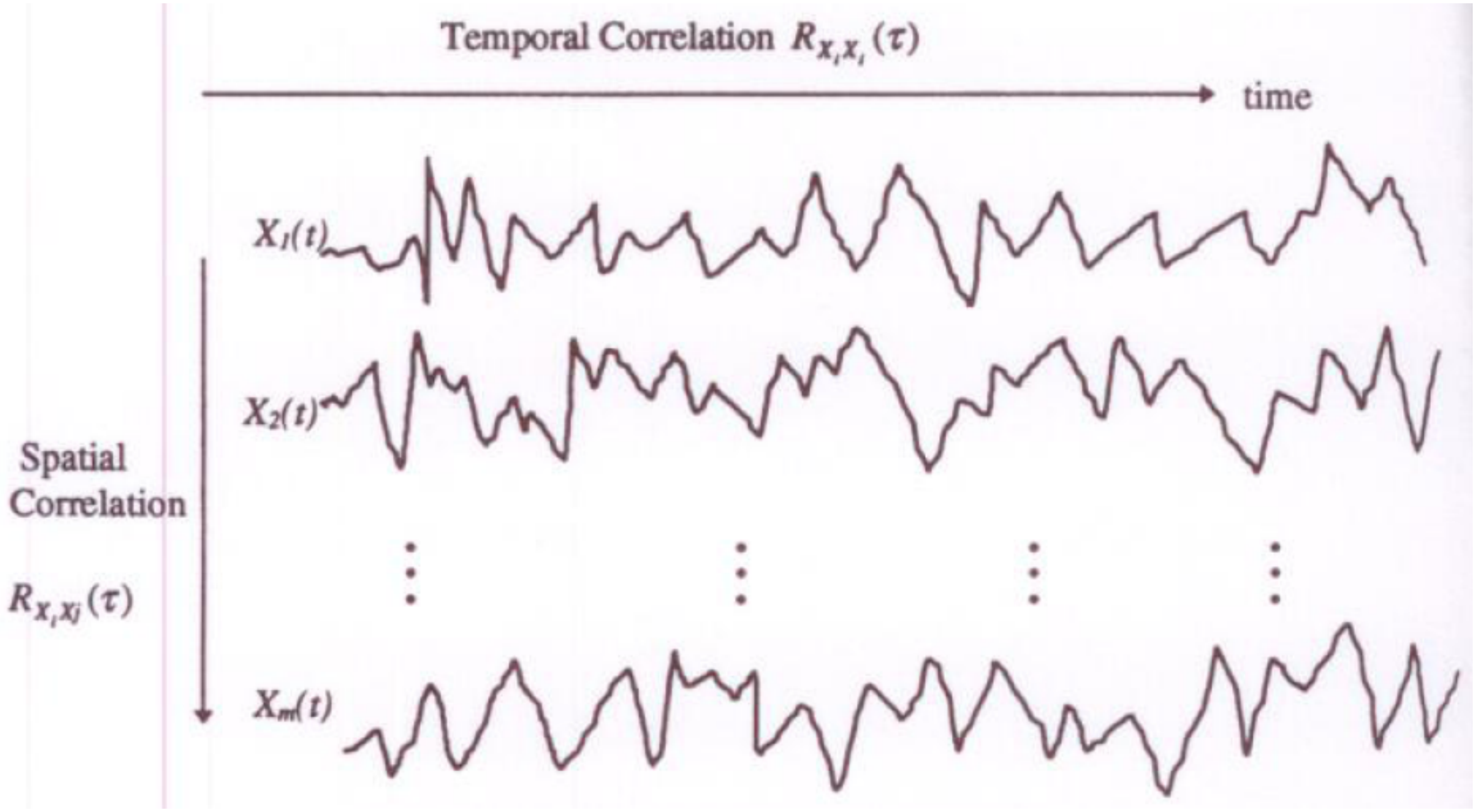
$$\boxed{\begin{aligned} X(kN+m) &= (2b_k-1)\text{pulse}(mT_s), \\ &\quad \text{for } k=\ldots,-2,-1,0,1,2,\ldots \quad \text{and } m=1,2,\ldots,N \end{aligned}}$$

They are the sampled values of a unit-amplitude pulse with a duration equal to the bit interval

The sampling rate is N times the bit rate

Correlated Random Processes

- Correlated vector-valued random processes



Correlated Gaussian Sequences: Scalar Case

- **Spectral Factorisation Method** – We can design a filter in the frequency domain to transform an uncorrelated Gaussian sequence into a correlated Gaussian sequence

- When we pass an independent Gaussian sequence through a filter, the output Gaussian process will have a PSD given by

$$\boxed{S_{YY}(f) = S_{XX}(f)|H(f)|^2}$$

where

$$\boxed{S_{XX}(f) = \sigma_X^2 \text{ for } |f| < \frac{1}{2}}$$

- Hence, if we choose $\sigma_X^2=1$, we have $S_{YY}(f)=|H(f)|^2$

Correlated Gaussian Vector Sequences

- We now consider the problem of generating sampled values of m zero-mean Gaussian random processes $Y_1(t), Y_2(t), \ldots, Y_m(t)$ with arbitrary PSDs and arbitrary correlation between them

❑ **Special Case** – We begin with the simpler case where the autocorrelation function of each of these processes is the same except for a scale factor and the cross-correlation function has the same functional form as the autocorrelation function

- The sampled values of these m scalar-valued processes can be treated as components of a vector-valued process

$$\boxed{\vec{\mathbf{Y}}(k) = \begin{bmatrix} Y_1(k) \\ Y_2(k) \\ \vdots \\ Y_m(k) \end{bmatrix}}$$

The covariance between the components of the process at a given time

The *temporal* correlation

with

$$\boxed{R_{Y_iY_j}(n) = \mathrm{E}\left[Y_i(k)Y_j(k+n)\right] = \sigma_{ij}R(n)}$$

Correlated Gaussian Vector Sequences

- We first generate a sequence of Gaussian vectors $\vec{\mathbf{X}}(k)$ with uncorrelated (and hence independent) Gaussian components where each component has the temporal correlation specified by R(n)
- This is done by generating m independent sequences (components of the vector $\vec{\mathbf{X}}$) with a given temporal correlation R(n)
- Given a random vector $\vec{\mathbf{X}}$ with a multivariate Gaussian distribution with mean zero and covariance matrix

$$\boxed{\mathrm{E}\left[\vec{\mathbf{X}}\vec{\mathbf{X}}^{\dagger}\right] \equiv \mathbf{\Sigma}_{\vec{\mathbf{X}}} = \mathbf{I}}$$

we want to transmit it into another Gaussian vector $\vec{\mathbf{Y}}$ with zero mean and covariance matrix

$$\boxed{\mathrm{E}\left[\vec{\mathbf{Y}}\vec{\mathbf{Y}}^{\dagger}\right] \equiv \mathbf{\Sigma}_{\vec{\mathbf{Y}}} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1m} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2m} \\ \vdots & & \ddots & \vdots \\ \sigma_{m1} & \sigma_{m2} & \cdots & \sigma_{mm} \end{bmatrix}}$$

Correlated Gaussian Vector Sequences

- This is achieved by a linear transformation of the form

$$\boxed{\vec{\mathbf{Y}} = \mathbf{A}\vec{\mathbf{X}}}$$

- This transforms the covariance matrix of $\vec{\mathbf{X}}$ which is $\mathbf{I}$ into

$$\boxed{\Sigma_{\vec{\mathbf{Y}}} = \mathbf{A}\Sigma_{\vec{\mathbf{X}}}\mathbf{A}^T = \mathbf{A}\mathbf{A}^T}$$

- The transformation $\mathbf{A}$ can be obtained by the Cholesky decomposition

❑ **General Case** – In the general case, the spatial and temporal correlation could be arbitrary and each component of $\vec{\mathbf{Y}}$ could have a different temporal correlation and the cross-correlation functions can also have different functional forms

- In this case, the process $\vec{\mathbf{Y}}$ is now specified by a sequence of covariance matrices, for each time lag j,

$$\boxed{\Sigma_{\vec{\mathbf{Y}}}(j)}$$

Pre-LAB 1 – BER Simulation

- A simple BPSK communication system can be written as

$$y(k) = \underbrace{P}_{\text{Signal power}} \times h(k)s(k) + n(k), \ \text{for time } k$$

where $y(k)$ is the received signal
$h(k)$ is the fading channel
$s(k)$ is the BPSK symbol $\in \{-1, +1\}$
$n(k)$ is the additive noise

- Detection is usually done by

$$\tilde{s}(k) = \begin{cases} +1, & \text{if } \mathrm{Re}\left\{\frac{y(k)}{h(k)}\right\} > 0, \\ -1, & \text{if } \mathrm{Re}\left\{\frac{y(k)}{h(k)}\right\} \leq 0, \end{cases}$$

Pre-LAB 1 – BER Simulation

- An error occurs if $\tilde{s}(k) \neq s(k)$
- BER is the probability that an error for detecting a bit occurs
- BER can be estimated by simulating the detection for **many** bits and counting the number of errors, normalised by the total number of bits
- Let’s go through the simulations together!

Pre-LAB 1 – BER Simulation in MATLAB

- Generate $s(k)$

```
⟩⟩ u=rand;
⟩⟩ if u>.5, s(k)=1; else s(k)=-1;
```

s(k) is either 1 or -1 with equal probability

- Generate $h(k)$

```
⟩⟩ h(k)=(randn+j*randn)/sqrt(2);
```

h(k) is complex Gaussian distributed, with 0 mean and 1 variance. Also, |h(k)| is Rayleigh distributed and ∠h(k) is uniformly distributed between 0 and 2π

- Generate $n(k)$

```
⟩⟩ n(k)=(randn+j*randn)/sqrt(2);
```

n(k) is complex Gaussian distributed, with 0 mean and 1 variance, or the noise power is 1

Pre-LAB 1 – BER Simulation in MATLAB

- Generate $y(k)$

```
⟩⟩ y(k)=P*h(k)*s(k)+n(k);
```

- Generate $\tilde{s}(k)$

```
⟩⟩ stilde(k)=(real(y(k)/h(k))>0)*1+(real(y(k)/h(k))<=0)*(-1);
```

- Count the number of errors

```
⟩⟩ if stilde(k)~=s(k), no_of_errors=no_of_errors+1;
```

Pre-LAB 1 – BER Simulation in MATLAB

- Note that we need to run the above for many k, i.e.,

```
〉〉 no_of_errors=0;
〉〉 for k=1:total_bits,
〉〉      Simulate stilde(k) to update no_of_errors
〉〉 end
```

- Calculate the BER

```
〉〉 BER=no_of_errors/total_bits;
```

- Then, we have obtained the BER for SNR=10log10(P) dB **ONLY**!

More Efficient MATLAB Simulations

- FOR LOOP is SLOW in MATLAB
- BUT can be avoided by using VECTORS/MATRICES
- For the BER simulations, we can have

```
〉〉 u=rand(total_bits,1);
〉〉 s(1:total_bits,1)=1; s(find(u<0.5))=-1;
〉〉 h=(randn(total_bits,1)+j*randn(total_bits,1))/sqrt(2);
〉〉 n=(randn(total_bits,1)+j*randn(total_bits,1))/sqrt(2);
〉〉 s_est=real((P*h.*s+n)./h);
〉〉 stilde(find(s_est>0))=1; stilde(find(s_est<=0))=-1;
〉〉 BER=sum(find(stilde~=s))/total_bits;
```