HW 6: Channel Capacity and Computational Complexity

ECE 5680 – Wireless Communication

- **Due date:** Monday, December 1, 2015 at 11:59pm
- **Submission instructions:** Upload your solutions (solutions and Matlab code) to Cornell Blackboard as a *single* `zip`-file; name the file ⟨netid⟩-hw⟨number⟩.zip. Include all simulation plots in either `pdf` or `eps` format in your archive. If you want to get detailed comments on your homework solutions, create a *single* `pdf`-file that contains all answers. I strongly suggest the use of LaTeX to typeset your responses, but a scanned version of your handwritten answers is also OK.
- **Total points:** 100pts

Problem 1: Capacity of a SISO Additive White Gaussian Noise Channel (34pts)

In this problem, we compute the channel capacity of a complex-valued SISO AWGN channel, and the maximum possible transmission rates for higher-order modulation schemes. We consider the following simple AWGN channel: $y = s + n$, where $y \in \mathbb{C}$ is the received signal, $s \in \mathcal{X}$ is the transmit signal with $\mathbb{E}_s\left[|s|^2\right] = E_s$, and $n \sim \mathcal{CN}(0, N_0)$ is AWGN. We learned that the capacity of this channel is given by

$$C_{\text{AWGN}} = \max_{P_x} I(X;Y) = \log_2(1 + SNR) \quad \text{[bits/s/Hz]}, \tag{1}$$

where P_x turns out to be a Gaussian distribution with variance E_s and $SNR = E_s/N_0$. This implies that the input distribution that achieves capacity in this channel is Gaussian and no rate R above C_{AWGN} enables reliable communication. What if we were forced to use BPSK, QPSK, or even a higher-order modulation schemes? What would the maximum transmission rate be at a certain SNR value, given we had some optimal coding scheme? This is what we are going to simulate next.

Part 1 Plot the channel capacity (1) in terms of bits/s/Hz dependent on the SNR (in decibel). Plot the SNR in the range from -10 dB to $+20$ dB, and use a linear scale for the y-axis.

Part 2 Imagine you were forced to use QPSK transmission but you can come up with an arbitrary coding scheme (e.g., something much more powerful than a Hamming code). What is the maximum possible rate you can achieve? Provide a brief explanation. *Hint: Do not try to come up with a coding scheme. Just think about what the maximum possible rate could be.*

Part 3 In the following parts of this problem, we will simulate the so-called *maximum achievable rate* given a modulation scheme such as BPSK or others. The maximum achievable rate is given by

$$C(SNR, \mathcal{X}) = I(X;Y), \tag{2}$$

i.e., the mutual information between the input signal $x \in \mathcal{X}$ and the output signal y, in the presence of AWGN. Similarly to the channel capacity, it is impossible to communicate reliably at a higher rate R than the achievable rate $C(SNR, \mathcal{X})$, given a particular modulation scheme $\mathcal{X}$.

In principle, one could analytically compute (2). For higher-order modulation schemes (such as 16-QAM), however, the resulting expressions are complicated. We will therefore perform Monte–Carlo simulations to compute the maximum achievable rate. It is important to realize that the mutual information $I(X;Y) = H(X) - H(X|Y)$ can be rewritten as

$$I(X;Y) = \int_{y \in \mathbb{C}} \sum_{s \in \mathcal{X}} p(x,y) \log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right) \mathrm{d}y,$$

which is equivalent to

$$I(X;Y) = \mathbb{E}_{y,x} \left[\log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right) \right]. \tag{3}$$

Hence, to approximate $I(X;Y)$, we can simply compute the *empirical* expected value of (3). More specifically, we can randomly generate input signals $x \in \mathcal{X}$, noise realizations $n \sim \mathcal{CN}(0, N_0)$, and compute $y = s + n$. We can then evaluate the $\log_2$-term in (3). By repeating this procedure multiple times, and by averaging the results, we can compute an empirical estimate of the expectation operation in (3).

Simplify the $\log_2$-term using Bayes' rule so that $p(x)$ vanishes. Furthermore, write out analytically the distribution $p(y)$. *Hint: The result will depend on the conditional probability $p(y|x)$ and $p(y)$, for which we know the distribution. The distribution $p(y)$ will contain a sum over all constellation points in $\mathcal{X}$.*

Part 4 Simulate the maximum achievable rate in (2) using the procedure outlined in Part 3 and your simplified $\log_2$-expression. Plot the maximum achievable rate for BSPK, QPSK, 16-QAM, and 64-QAM in terms of bits/s/Hz dependent on the SNR (in decibel). Perform a sufficiently large number of Monte–Carlo trials to get smooth curves for the achievable rates. Also include the capacity from Part 1. Plot the SNR in the range from $-10\,\mathrm{dB}$ to $+20\,\mathrm{dB}$, and use a linear scale for the y-axis.

Part 5 In Part 2, you were asked to guess the maximum achievable rate for QPSK transmission. Does the simulation result from Part 4 confirm your answer in Part 2?

Remark: As you will see, at sufficiently low SNR values, the use of BSPK, QPSK, 16-QAM, and 64-QAM are optimal, i.e., they achieve the same rates as the channel capacity C_{AWGN}, which require a Gaussian codebook. Hence, the use of strong codes in combination with these modulation schemes will perform as good as AWGN codebooks, which we know are not practical.

Problem 2: Capacity in a MIMO Rayleigh Fading Channel (33pts)

In this problem, we will simulate the ergodic and ε-outage capacities in a flat Rayleigh fading system with equal power per transmit antennas. The considered MIMO system is equipped with M_R receive antennas and M_T transmit antennas. The input-output relation is $\mathbf{y} = \mathbf{Hs} + \mathbf{n}$, where $\mathbf{y} \in \mathbb{C}^{M_R}$ is the receive vector, $\mathbf{H} \in \mathbb{C}^{M_R \times M_T}$ the channel matrix, $\mathbf{s} \in \mathcal{X}^{M_T}$ the transmit data vector (with $\mathcal{X}$ denoting the constellation set), and additive Gaussian noise $\mathbf{n} \sim \mathcal{CN}(\mathbf{0}, \mathbf{I}N_0)$.

Part 1 Instead of writing your own capacity simulator (which is now slightly more complicated than the simulator from Problem 1), download the MATLAB simulator from

```
http://www.csl.cornell.edu/~studer/lectures/ECE5680/cap_sim_fixthis.m
```

and simulate the ergodic capacity vs. SNR curves (as in Problem 1) for the following scenarios:

- $M_T = 1$, $M_R = 1$
- $M_T = 1$, $M_R = 4$
- $M_T = 4$, $M_R = 1$
- $M_T = 4$, $M_R = 4$

and generate a single plot that includes all four capacity curves. Explain what you see in the plot. Do the curves make sense? *Hint: In this part, ignore the results for the achievable rate.*

Part 2 Now, fix the system parameters to $M_T = M_R = 4$ and simulate the maximum achievable rates for BSPK, QPSK, and 16-QAM. Also include the channel capacity (assuming a Gaussian codebook) in your plot. Increase the y-axis range to 20 bits/channel use. Do you think that at sufficiently low SNR values, the use of BPSK, QPSK, and 16-QAM is optimal?

Part 3 The simulations carried out in Parts 1 and 2 are for fast fading systems, i.e., we compared the ergodic channel capacity and ergodic maximum achievable rate. We now consider a flat fading system, where the channel remains constant for an entire code block. To this end, we are interested in the so-called ε-outage capacity $C_{\varepsilon,\text{out}}$ defined as

$$\Pr[C(\mathbf{H}) \leq C_{\varepsilon,\text{out}}] = \varepsilon.$$

In the capacity simulator you downloaded before, we already compute the individual channel capacity results $C(\mathbf{H})$ (as well as the achievable rates) for every individual channel realization and store it in the following structures:

```
res.GCAP % contains Gaussian capacity results for channel matrices x SNR values
res.DCAP % contains achievable rate results for channel matrices x SNR values
```

We then compute the average across matrices to obtain the ergodic capacity for every SNR value. Have a close look how the simulator works, i.e., how exactly the capacity results are stored and computed. Now use both of these results to extract the ε-outage capacity. Fix $\varepsilon = 10\%$ and compute the ε-outage capacity for the same scenarios as in Parts 1 and 2. Generate two separate plots. *Hint: The exact location where you should add your code is clearly marked in the simulator.*

Problem 3: Complexity of the Zero Forcing MIMO Detector (33pts)

In Homework 5, you simulated a large number of MIMO data detectors. In this problem, you are supposed to compute the number of real-valued multiplications required to perform zero-forcing (ZF) data detection. Use the "simple MIMO simulator" from `http://www.csl.cornell.edu/~studer/software_mimo.html`. Create a new ZF detector function that performs the following steps:

1. Compute the matched filter: $\mathbf{y}^{\text{MF}} = \mathbf{H}^H\mathbf{y}$
2. Compute the Gram matrix: $\mathbf{G} = \mathbf{H}^H\mathbf{H}$
3. Compute the inverse of the Gram matrix $\mathbf{G}^{-1}$
4. Compute $\mathbf{v} = \mathbf{G}^{-1}\mathbf{y}^{\text{MF}}$
5. Perform element wise detection: $\hat{s}_i^{\text{ZF}} = \arg\min_{s\in\mathcal{X}} |v - s|^2, i = 1, \ldots, M_T$

For each of these steps, separately compute the computational complexity (in real-valued multiplications). For every complex-valued multiplication, assume that you need four real-valued multiplications, if you have to compute $|x|^2 = x_R^2 + x_I^2$ you will need two real-valued multiplications, if you have to compute αx, where $\alpha \in \mathbb{R}$, you need two real-valued multiplications, etc. In what follows, use the variables

```
par.MR % receive antennas
par.MT % transmit antennas
```

to obtain generic multiplication counts, i.e., whenever you change your simulation parameters, you automatically get the correct multiplication counts.

Part 1 Compute the number of multiplications that are needed to compute $\mathbf{y}^{\text{MF}} = \mathbf{H}^H\mathbf{y}$ in Step 1.

Part 2 Compute the number of multiplications that are needed to compute $\mathbf{G} = \mathbf{H}^H\mathbf{H}$ in Step 2. *Hint: remember that the Gram matrix is symmetric, i.e., exploit symmetry to reduce the number of required multiplications*

Part 3 ***This is the hard part!*** There are many different methods to perform matrix inversion. We will use one of the simplest methods, namely the Cholesky decomposition (Wikipedia has a pretty decent explanation of the algorithm). The main idea of this approach is to first decompose the Gram matrix $\mathbf{G} = \mathbf{L}\mathbf{L}^H$ into a product of a lower triangular matrix $\mathbf{L}$ and its Hermitian transpose. Since $\mathbf{G}^{-1} = \mathbf{L}^{-H}\mathbf{L}^{-1}$, we first compute $\mathbf{L}^{-1}$ and then compute the product $\mathbf{L}^{-H}\mathbf{L}^{-1}$. The inverse $\mathbf{L}$ can be computed efficiently using the following procedure: Define the matrix $\mathbf{L}^{-1} = \mathbf{A}$. From this, we have $\mathbf{I} = \mathbf{L}\mathbf{A}$. Each column in $\mathbf{I}$ is an unit vector $\mathbf{e}_i$ where all entries are zero except the ith entry is one. We can now compute every column $\mathbf{a}_i$ from the matrix $\mathbf{A}$ separately. Consider $\mathbf{e}_i = \mathbf{L}\mathbf{a}_i$, where $\mathbf{e}_i$ is given and $\mathbf{a}_i$ (the ith row of $\mathbf{A}$) must be computed. This can be accomplished by using *back-substitution* (if you have not heard of it, the Internet will be your friend). Back substitution successively finds entries of the matrix $\mathbf{a}_i$ starting from the first entry and proceeding to the last. After obtaining $\mathbf{L}^{-1}$ compute $\mathbf{G}^{-1} = \mathbf{L}^{-H}\mathbf{L}^{-1}$. Compute the number of real-valued multiplications that are required to compute $\mathbf{G}^{-1}$ using this procedure. *Hint: I suggest you to first implement the above mentioned algorithm to see whether it really works (See also Part 7). For the sake of simplicity, ignore any structure in the vectors, even if there are a lot of zeros which would not need to be multiplied.*

Remark: As you have just experienced, even very common MATLAB functions such as `inv(A)` may turn out to be extremely complicated and annoying. It is very common that researchers underestimate the complexity of an algorithm when working with MATLAB only. Don't be one of these!

Part 4 Compute the number of real-valued multiplications that are needed to perform Step 4.

Part 5 Compute the number of real-valued multiplications that are needed to perform Step 5. *Hint: This step will depend on the modulation order*.

Part 6 Consider a $M_T = M_R = 4$ MIMO system with 16-QAM. Use the results from Parts 1, 2, 3, 4, and 5 to compute a bar plot for the number of multiplications required in every step. Which of the above steps is the most complex one?

Remark: Operation counts are a common tool to get first-order estimates of an algorithm's computational complexity. In practice, however, other factors such as the amount of parallelism, the regularity of the algorithm, the numerical stability, the memory requirements, etc. may be even more important.

Part 7 If you have not done it already, Implement the entire procedure as detailed above in MATLAB and compare the bit error rate to the ZF detector that is already in the simulator. Provide a plot. *Hint: You should get exactly the same results!*