SPECTRE
Short Packet Communication Toolbox

Version 0.2

Contributors (alphabetic order):
Austin Collins, Giuseppe Durisi, Tomaso Erseghe, Victoria Kostina,
Johan Östman, Yury Polyanskiy, Ido Tal, Wei Yang

e-mail: fblcode-list@mit.edu

July 25, 2016

Abstract

Shannon theory describes fundamental limits of communication and compression systems. Classic closed-form results (such as the well known $\log(1 + \mathrm{SNR})$ formula) apply only to the regime of infinite blocklength (infinite packet size/delay). For finite blocklengths, no closed-form results are usually obtainable, but there exist tight upper and lower bounds, as well as approximations. This manual describes numerical routines for computing these bounds and approximations for some popular channel and source models.

The toolbox is under development and the participation of additional members of the information and communication theory communities in this endeavor is warmly welcomed!

Contents

Chapter 1

Motivation

In his 1948 landmark paper, Claude E. Shannon demonstrated that communication with arbitrarily small probability of error is feasible if, and only if, the communication rate is below the so called channel capacity. To achieve this result, codes with large packet size (blocklength) must be employed. Shannon's channel capacity has served as a useful guideline to design wireless communication systems.

In some applications, however, a more refined analysis of the interplay between packet-error probability, communication rate, and packet size is required. This may occur in emerging applications, such as massive machine-to-machine communication for metering, traffic safety, and telecontrol of industrial plants, together with real-time data transfer to enable remote wireless control (tactile internet), which may require the exchange of short packets, sometimes under stringent latency and reliability constraints.

Finite-blocklength information theory, a subfield of information theory that benefitted from seminal contributions from Shannon, Strassen, and Dobrushin, among others, is currently a very active field of research. Nonasymptotic achievability and converse bounds on the *maximum coding rate* achievable for a given packet size and packet error probability are now available for several channel models that are relevant for wireless communication systems, such as the AWGN channel, the quasi-static Rayleigh fading channel, and the Rayleigh block-fading channel. The purpose of this toolbox is to provide numerical routines for the computation of these bounds. In the next chapters, we provide a bare-bone manual, which describes the routines available in this toolbox.

Chapter 2

The real AWGN channel

General info:

- Maintainer: Y. Polyanskiy <yp@mit.edu>
- Main references: [1]
- Example: See Fig. 2.1
- Channel model:
$$Y_j = X_j + Z_j, \quad j = 1, \ldots, n$$
  - Codewords are real: $x_j \in \mathbb{R}, j = 1, \ldots, n$
  - Noise: $Z_j \sim \mathcal{N}(0, 1)$ iid.
  - Power constraints: Each codeword x^n satisfies
$$\sum_{j=1}^{n} |x_j|^2 \leq nP$$
- Common input/output arguments:
  1. `P` – input argument; parameter of the power constraint (so $P = 10$ is $SNR = 10$ dB).
  2. `epsil` – block probability of block error.
  3. `lm` or `Lms` – output argument; $\log_2 M$, log-size of codebook (base-2 information units).
  4. `n` – input argument; blocklength.
     For slow functions that do not support vectorized arguments.
  5. `Ns` – input argument; vector of blocklengths.

2.1 Summary plot: `plot_v3()`

Definition:

```
function [Ns lb ub feinst gal wlfz] = plot_v3(P, epsil)
```

This function plots various bounds on the same figure. See Fig. 2.1.

```
> plot_v3(10, 0.001, 1, 100:20:1000)
```

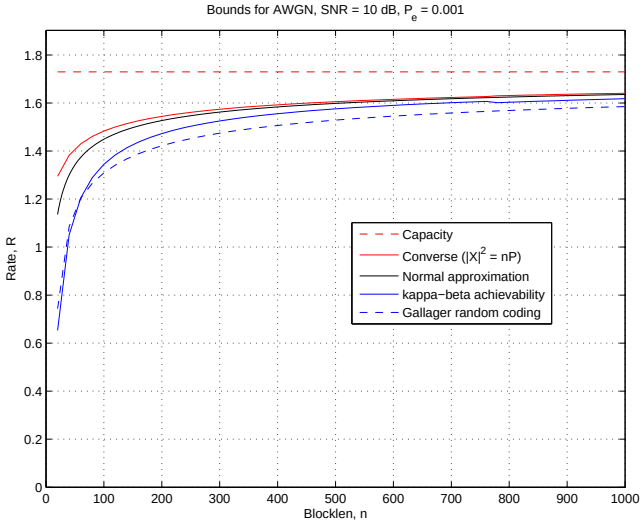

Figure 2.1: Code and resulting picture for AWGN bounds

2.2 Achievability: `shannon_ach2()`

Definition:

```
function lm = shannon_ach2(n, epsil, P)
```

Function computes Shannon's cone-packing achievability bound, see [1, (41)].

2.3 Achievability: `kappabeta_ach()`

Format:

```
function Lms = kappabeta_ach(Ns, epsil, P, hack)
```

This computes the $\kappa\beta$ achievability bound, see left-hand inequality in [1, (218)]. If $hack \neq 0$, then we use asymptotic approximation for κ. This increases the speed dramatically and is very precise already at $n = 10$, see [1, (217)]. If *hack* is not set, it defaults to 1.

2.4 Achievability: `gallager_ach()`

Format:

```
function lm = gallager_ach(n, epsil, P)
```

Computes Gallager's achievability bound, see [1, (44)].

2.5 Converse: converse()

Format:

```
function Lms = converse(Ns, epsil, P)
```

This computes the meta-converse lower bound with $Q_{Y^n} = \mathcal{N}(0, 1+P)^n$, see right-hand inequality in [1, (218)].

2.6 Normal approximation: normapx_awgn(), normapx_biawgn()

Format:

```
function Lms = normapx_awgn(Ns, epsil, P);
```

Fast and frequently very precise approximation to both achievability and converse:

$$\log M \approx nC - \sqrt{nV}Q^{-1}(\epsilon) + \frac{1}{2}\log n$$

2.7 Code database: plot_universe()

Code in `universe/plot_universe()` compiles a comparative plot of normalized rate for different codes. See [1, Section IV.D].

Chapter 3

Block-memoryless Rayleigh fading channel, no CSI

General info:

- Maintainer: G. Durisi <durisi@chalmers.se>
- Contributors: G. Durisi, J. Östman, W. Yang
- Main reference: [2]
- Example: See Fig. 3.1.
- Channel model: Rayleigh block-fading channel with m_{t} transmit antennas and m_{r} receive antennas that stays constant for n_{c} channel uses. The blocklength n is an integer multiple of the coherence time n_{c}, i.e., $n = ln_{\mathrm{c}}$; here l denotes the number of independent time-frequency branches:

$$\mathbb{Y}_k = \mathsf{X}_k \mathbb{H}_k + \mathbb{W}_k\,, \qquad k = 1, \ldots, l\,, \tag{3.1}$$

  where $\mathbb{Y}_k, \mathbb{W}_k \in \mathbb{C}^{n_{\mathrm{c}} \times m_{\mathrm{r}}}$, $\mathsf{X}_k \in \mathbb{C}^{n_{\mathrm{c}} \times m_{\mathrm{t}}}$ and $\mathbb{H}_k \in \mathbb{C}^{m_{\mathrm{t}} \times m_{\mathrm{r}}}$. The noise $\mathbb{W}_k$ and the channel gain $\mathbb{H}_k$ are independent and have independent $\mathcal{CN}(0,1)$ entries. Neither the transmitter nor the receiver are aware of the realizations of the fading matrix $\mathbb{H}_k$.

  The power constraint is defined *per coherence interval*, i.e., each codeword C consists of l subcodewords $\mathsf{C}_1, \ldots, \mathsf{C}_l$ that must satisfy:

$$\mathrm{tr}\{\mathsf{C}_k^H \mathsf{C}_k\} = n_{\mathrm{c}}\rho, \quad k = 1, \ldots, l.$$

  Here, ρ denotes the SNR.

- Common input, output arguments:
  - `snrdB` – SNR in dB
  - `T` – channel coherence time (expressed in channel uses)
  - `L` – number of independent fading realizations spanned by each codeword
  - `Mt` – number of tranmsmit antennas

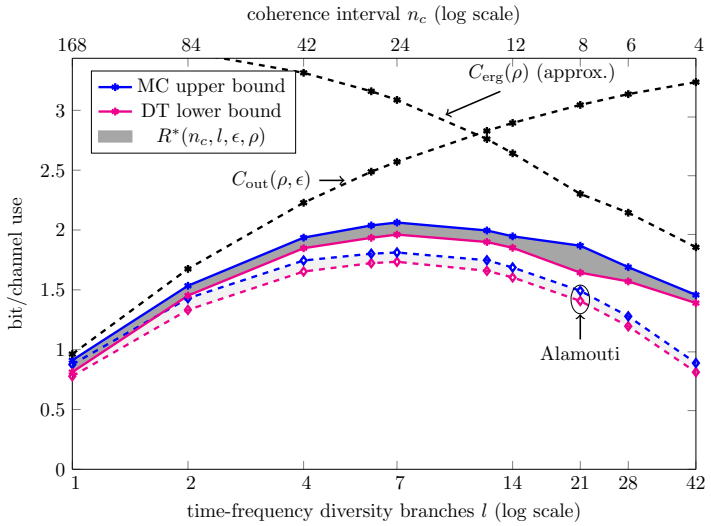

Figure 3.1: Maximum coding rate $R^*(n_c, l, \epsilon, \rho)$ for 2×2 MIMO system operating over a Rayleigh block-fading channel with coherence time n_c. Each packet spans l coherence intervals. The packet size is 168 channel uses. The packet error rate is $\epsilon = 10^{-3}$. The SNR is $\rho = 6$ dB.

- `Mr` – number of receve antennas
- `epsilon` – packet error rate (block error rate)
- `prec` – $\log_2$ of the number of samples used in the Monte-Carlo step; this parameter should be set so that $2^{\texttt{prec}} \gg 100/\texttt{epsilon}$
- `filename` – data file in which the samples of the information density are saved for possible future refinements (provided that the flag `SAVE` is set)
- `R` – estimate of the maximum coding rate
- `current_eps` – actual bound on the packet error rate (it may deviate from `epsilon` if `prec` is not chosen appropriately)

3.1 Achievability bounds: `DT_USTM_NxM()`, `DT_USTM_MxM()`

Format:

```
function [R,current_eps]
=DT_USTM(snrdB,T,L,Mt,Mr,epsilon,prec,filename);
```

This function computes the USTM-DT achievability bound [2, Th. 3] for the case $m_{\text{t}} \leq m_{\text{r}}$.

3.2 Converse helper functions: `MC_USTM_2x2_q1x2()`, `MC_USTM_2x2_q_2x2()`, `MC_USTM_4x4_q_Mx4`

Format

```
function [R,current_eps,current_prec]
=MC_USTM_2x2(snrdB,T,L,Mtalt,epsilon,prec,pow_all,filename)
function [R,current_eps,current_prec]
=MC_USTM_4x4(snrdB,Mtalt,T,L,epsilon,prec,pow_all,filename)
```

These two functions compute the right-hand side of [2, Eq. (41)] for fixed input diagonal matrices $\{\Sigma_k\}_{k=1}^{l}$ and fixed auxiliary output distribution (which is parameterized by the number of effectively used transmit antennas `Mtalt` that result in this distribution), for the case $m_{\mathrm{t}} = m_{\mathrm{r}} = 2$ and $m_{\mathrm{t}} = m_{\mathrm{r}} = 4$. To obtain the actual converse bound, a further optimization over these parameters need to be performed. This is discussed in the next section. The parameter `pow_all` is related to the diagonal entries of the matrices $\{\Sigma_k\}_{k=1}^{l}$. For the $m_{\mathrm{t}} = m_{\mathrm{r}} = 2$ case, `pow_all` is an l-dimensional vector, whose entries are in $[0, 0.5]$. Indeed, the particular formulation of the power constraint in [2, Eq. (41)] implies that the other entry is uniquely determined from the first one. For the 4×4 case, the code provided assumes that the optimization in [2, Eq. (41)] is restricted only to $\{\Sigma_k\}_{k=1}^{l}$ of the form $\Sigma_1 = \cdots = \Sigma_l = \Sigma$. In this case, `pow_all` is a 4-dimensional vector that contains the diagonal entries of Σ.

3.3 Converse bound: `compute_MC_2x2_telatar`

Format

```
function R=compute_MC_2x2_telatar(snrdB,T,L,epsilon)
```

Compute the metaconverse upper bound [2, Eq. (41)] by invoking the function `MC_USTM_2x2()` and by optimizing over the input diagonal matrices $\{\Sigma_k\}_{k=1}^{l}$ and the number of effectively used transmit antennas. The optimization over the $\{\Sigma_k\}_{k=1}^{l}$ is restricted to 2×2 diagonal matrices whose diagonal entries are either [`SNR`/2, `SNR`/2] or [`SNR`, 0] according to the generalized form of Telatar conjecture reported in [2, Eq. (42)]. NOTE: this function may be very slow if `L` is large or `epsilon` is small.

3.4 Alamouti: `DT_USTM_Alamouti()`, `MC_USTM_Alamouti()`

Format

```
function [R,current_eps]
=MC_USTM_Alamouti(snrdB,T,L,epsilon,prec,filename)
function [R,current_eps]
=DT_USTM_Alamouti(snrdB,T,L,epsilon,prec,filename)
```

These functions compute achievability and converse bounds for a 2×2 MIMO system with Alamouti used as inner code

3.5 Outage: `outage_mi()`, `outage_alamouti()`, `outage_2x2_telatar()`

```
function [Cout,current_eps]
=outage_mi(snrdB,epsilon,Mt,Mr,L,pow_all,prec)
```

```
function [Cout,current_eps]
=outage_alamouti(snrdb,epsilon,L,prec,filename)
function R=outage_2x2_telatar(L,prec,epsilon,snrdb)
```

The first two functions compute the outage mutual information I_ϵ for a given set of diagonal input covariance matrices, and the outage mutual information with Alamouti, respectively:

$$\mathbb{P}[\mathbb{E}[\imath(X^l;\mathbb{Y}^l)|\mathbb{H}^l] \leq I_\epsilon] = \epsilon\,,$$

where in the first case, the covariance structure of $X^l = (X_1, \ldots, X_l)$ is specified by `pow_all` and

$$\imath(X^l;\mathbb{Y}^l) = \sum_{k=1}^{l} \log \frac{dP_{\mathbb{Y}_k|X_k}}{dP_{\mathbb{Y}_k}}\,.$$

The `outage_mi` function evaluates the outage mutual information corresponding to a given input covariance matrix, whose eigenvalues are specified in `pow_all`. Specifically, `pow_all` is a `Mt-1 x L` matrix containing the first `Mt`-1 eigenvalues of the input covariance matrices corresponding to each coherence interval. The last eigenvalue follows from the power constraint. To obtain the outage capacity one has to optimize over all possible choices of `pow_all`.

The function `outage_2x2_telatar` provides an example on how to perform this optimization for the 2×2 MIMO case. The optimization is restricted to input covariance matrices that satisfy the generalized Telatar conjecture [2, Eq. (42)] according to which in the eigenvalues of the input covariance matrix in each coherence interval satisfy the following property: k out the `Mt` eigenvalues are equal to `SNR`$/k$ and the remaining `Mt`$-k$ eigenvalues are equal to zero. The value of k may change across the coherence intervals.

3.6 Ergodic capacity: `ergodic_USTM()`

```
function RergUSTM=ergodic_USTM(snrdB,T,Mt,Mr,prec)
```

This function computes a lower bound on the ergodic capacity (no CSIR), obtained by evaluating the mutual information for a USTM input distribution. It is assumed that $m_{\mathrm{t}} \leq m_{\mathrm{r}}$.

Chapter 4

Quasi-static fading channel

General info:

- Maintainer: W. Yang <weiy@princeton.edu>
- Main references: [3]
- Example: See Fig. 6.1
- Channel model: a quasi-static MIMO fading channel with m_{t} transmit and m_{r} receive antennas. The channel input-output relation within n channel uses is given by

$$\mathsf{Y} = \mathsf{X}\mathbb{H} + \mathbb{Z} \tag{4.1}$$

  where $\mathsf{X} \in \mathbb{C}^{n \times m_{\mathrm{t}}}$ is the signal transmitted over n channel uses; $\mathbb{Y} \in \mathbb{C}^{n \times m_{\mathrm{r}}}$ is the corresponding received signal; the matrix $\mathbb{H} \in \mathbb{C}^{m_{\mathrm{t}} \times m_{\mathrm{r}}}$ contains the complex fading coefficients, which are random but remain constant over the n channel uses; $\mathbb{Z} \in \mathbb{C}^{n \times m_{\mathrm{r}}}$ denotes the additive noise at the receiver, which is independent of $\mathbb{H}$ and has iid $\mathcal{CN}(0, 1)$ entries. Power constraints: each codeword x^n satisfies

$$\|\mathsf{X}\|_{\mathrm{F}}^2 \leq nP.$$

  In the code, we also assume that the entries of $\mathbb{H}$ are iid Rician-distributed, i.e.,

$$H_{ij} \sim \mathcal{CN}(\sqrt{K/(K+1)}, 1/(K+1))$$

  where K is the Rician K-factor.

- The code for the MIMO case ($m_{\mathrm{t}} > 1$) is built under the additional assumption that all codewords X are "isotropic", i.e., (see [3, Sec. III])

$$\frac{1}{n}\mathsf{X}^{\mathrm{H}}\mathsf{X} = \frac{P}{m_{\mathrm{t}}}\mathsf{I}_{m_{\mathrm{t}}}. \tag{4.2}$$

  Currently, we do not have code for MIMO achievability and converse bounds that hold without the assumption (4.2).

- Input/output arguments:

  1. `nn` – input argument; vector of blocklengths.

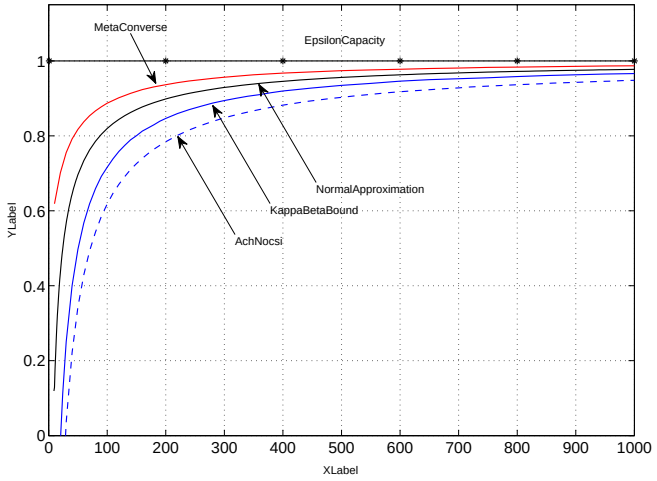

Figure 4.1: Achievability and converse bounds for a quasi-static SIMO Rician-fading channel with K-factor equal to 20 dB, two receive antennas, SNR $= -1.55$ dB, and $\epsilon = 10^{-3}$.

2. `error` – input argument; block error probability.
3. `tx`, `rx` – input arguments; number of transmit and receive antennas.
4. `P` – input argument; parameter of the power constraint (linear scale).
5. `K` – input argument; Rician K-factor. If `K` is not set, it defaults to 0, in which case the entries of $\mathbb{H}$ follow a Rayleigh distribution.
6. `rate_a`, `rate_c`, `rate_na` – output arguments; lower (achievability) bound, upper (converse) bound, and normal approximation of $(\log_2 M)/n$, channel coding rate (base-2 information units).

4.1 Summary plot: `plot_all()`

Format:

```
[C_e, rate_a, rate_c, rate_na] = plot_all(nn, P, error, tx, rx, K);
```

This function plots capacity/achievability/converse/normal approximation curves on the same figure. See Fig. 6.1.

4.2 Achievability: `ach_simo_csir()`

Format:

```
function rate_a = ach_simo_csir(nn, P, error, rx, K);
```

This function computes the $\kappa\beta$ achievability bound over a single-input multiple-output (SIMO) channel with perfect CSIR, see [3, Footnote 7].

4.3 Achievability: ach_simo_nocsi()

Format:

```
function rate_a = ach_simo_nocsi(nn, P, error, rx, K);
```

This function computes the achievability bound over a SIMO channel with no CSI, see [3, Eq. (67)].

4.4 Achievability: mimo_iso_ach()

Format:

```
function rate_a = mimo_iso_ach(nn, P, error, tx, rx, K);
```

This function computes the achievability bound over a MIMO channel with no CSI, where all input codewords satisfy $\mathsf{X}^{\mathrm{H}}\mathsf{X} = \frac{nP}{m_{\mathrm{t}}}\mathsf{I}_{m_{\mathrm{t}}}$, see [3, Eq. (65)].

4.5 Converse: converse_simo()

```
function rate_c = converse_simo(nn,P, error, rx, K);
```

This function computes the converse bound over a single-input multiple-output (SIMO) channel with CSIRT, see [3, Eq. (76)].

4.6 Converse: mimo_iso_conv()

```
function rate_c = mimo_iso_ach(nn, P, error, tx, rx, K);
```

This function computes the converse bound over a MIMO channel with CSIRT, where all input codewords satisfy $\mathsf{X}^{\mathrm{H}}\mathsf{X} = \frac{nP}{m_{\mathrm{t}}}\mathsf{I}_{m_{\mathrm{t}}}$, see [3, Eq. (78)].

4.7 Normal approximation: normapprox_simo(), normapprox_mimo_iso()

Format:

```
function rate_na = normapprox_simo(nn,P, error, rx, K)
function rate_na = normapprox_mimo_iso(nn, P, error, tx, rx, K);
```

These two functions compute the normal approximation $R^{\mathcal{N}}(\epsilon, n)$ to both achievability and converse, obtained by solving the following equation

$$\epsilon = \mathbb{E}\left[Q\left(\frac{C(\mathbb{H}) - R^{\mathcal{N}}(\epsilon, n)}{\sqrt{nV(\mathbb{H})}}\right)\right]$$

where

$$C(\mathsf{H}) \triangleq \log_2 \det\left(\mathsf{I}_{m_{\mathrm{t}}} + \frac{P}{m_{\mathrm{t}}}\mathsf{H}\mathsf{H}^{H}\right)$$

and

$$V(\mathsf{H}) \triangleq m \log_2^2 e - \sum_{i=1}^{m} \frac{\log_2^2 e}{(1 + P\lambda_i/m_{\mathrm{t}})}$$

with $m \triangleq \min\{m_{\mathrm{t}}, m_{\mathrm{r}}\}$ and $\{\lambda_i\}$ being the eigenvalues of $\mathsf{H}\mathsf{H}^{\mathrm{H}}$.

Chapter 5

Quasi-static fading channel (long-term power constraint)

General info:

- Maintainer: W. Yang <weiy@princeton.edu>
- Main references: [4]
- Channel model:

$$Y_i = HX_i + Z_i, \qquad i = 1, \dots, n \tag{5.1}$$

  where

  - $X_i, Y_i \in \mathbb{C}$ are the channel input and output, respectively
  - $H \in \mathbb{C}$ is the complex channel coefficient
  - $Z_i \sim \mathcal{CN}(0, 1)$ is the i.i.d. white Gaussian noise

- An $(n, M, \epsilon)_{\text{lt}}$ code for the channel (5.1) consists of
  - an encoder $f : \{1, \dots, M\} \times \mathbb{C} \to \mathbb{C}^n$ that maps the message $W \in \{1, \dots, M\}$ and the channel coefficient H to a codeword $x^n = f(W, H)$ satisfying the *long-term* (i.e., averaged-over-all-codeword) power constraint

$$\mathbb{E}_{W,H}\left[\|f(W, H)\|^2\right] \le nP \tag{5.2}$$

    where W is equiprobable on $\{1, \dots, M\}$
  - a decoder $g : \mathbb{C}^n \times \mathbb{C} \to \{1, \dots, M\}$ satisfying the average error probability constraint

$$\mathbb{P}[g(Y^n, H) \neq W] \le \epsilon \tag{5.3}$$

    where Y^n is the channel output induced by the transmitted codeword $x^n = f(W, H)$ according to (5.1).

- Remark: for comparison, a code is said to satisfy a *short-term* (i.e., per-codeword) power constraint if the encoder satisfies

$$\|f(W, H)\|^2 \le nP \tag{5.4}$$

  with probability one, see Chapter 4.

- Common input/output arguments:
  - `snr_db` – input argument; SNR in dB
  - `nn` – input argument; blocklength
  - `error` – input argument; block error probability
  - `rate_ach_lt`/`rate_conv_lt`/`rate_na_lt` – output arguments; lower (achievability) bound, upper (converse) bound, and normal approximation of $(\log_2 M)/n$, maximal achievable rate (in bits/ch. use).

5.1 The AWGN channel ($H = 1$)

Format

```
function rate_ach_lt=awgn_ach_lt(snr_db, nn, error);
function rate_conv_lt=awgn_conv_lt(snr_db, nn, error);
function rate_na_lt = awgn_na_lt(snr_db,nn,error);
```

These functions compute achievability/converse/normal approximation of $(\log_2 M)/n$, maximal achievable rate (in bits/ch. use). The bounds are from [4, Section II.C] .

5.2 Quasi-static fading channel with perfect CSIT

Under construction.

Chapter 6

Block-memoryless Rayleigh Fading (CSIR)

General info:

- Maintainer: A. Collins <austinc@mit.edu>
- Main references: Coming Soon
- Example: See Figure 6.1
- Channel Model: with n_t transmit antennas, n_r receive antennas, and coherence time T, the model is given by

  $$Y_j = H_j X_j + Z_j, \qquad j = 1, \ldots, n$$

  Where $\mathbb{F}$ is either $\mathbb{R}$ or $\mathbb{C}$, and

  - $X_j \in \mathbb{F}^{n_t \times T}$ is channel input
  - $H_j \in \mathbb{F}^{n_r \times n_t}$ is fading matrix, iid Rayleigh per block
  - $Z_j \in \mathbb{F}^{n_r \times T}$ is the i.i.d. (circularly symmetric) Gassian noise matrix.
  - $Y_j \in \mathbb{F}^{n_r \times T}$ is output matrix

  The input must satisfy the per-codeword power constraint

  $$\sum_{j=1}^{n} \|x_j\|_F^2 \leq nP$$

- Input / output arguments
  - `P` – Power constraint
  - `n` – number of coherent blocks (for nT total channel uses)
  - `n_t` – number of transmitter antennas
  - `n_r` – number of receiver antennas
  - `T` – channel coherence time (in number of channel uses)
  - `epsilon` – average probability of error
  - `real_or_complex` – specifies real or complex channel (set to `'real'` or `'complex'`, respectively)

6.1 Summary plot: `plot_all()`

Format:

```
function log_M = plot_all(P,n_t,n_r,T,epsilon)
```

This will plot the achievability bound, the normal approximation, and the capacity as in Figure 6.1. To change the plot range, the `n_vec` vector can be changed to include the desired points.

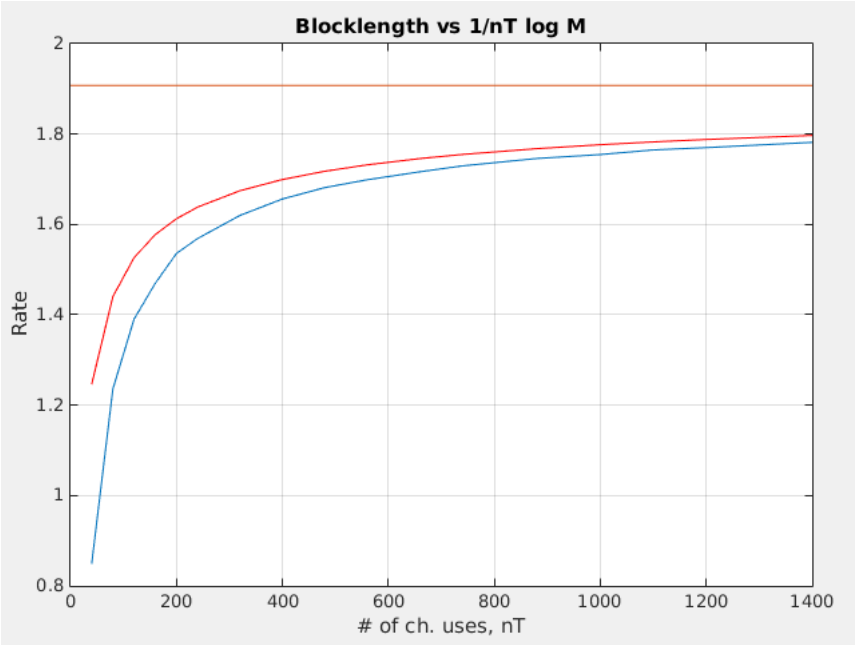

Figure 6.1: Achievability (blue), normal approximation (red), and capacity for the real MIMO CSIR block fading channel with $n_t = n_r = T = 2$, $P = 10$, $\epsilon = 10^{-3}$.

6.2 Achievability Bound: `MIMO_achievability_CSIR()`

Format:

```
function log_M = MIMO_achievability_CSIR(n,epsilon,n_t,n_r,T,P,real_or_complex)
```

This function computes the following average probability of error achievability bound:

$$\log M \geq \sup_{0<\tau<\epsilon} \sup_{Q_Y} \frac{\beta_\tau(P_Y, Q_Y)}{\beta_{1-\epsilon+\tau}(P_{XY}, P_X Q_Y)}$$

Where P_Y is the output distribution through the channel when the input P_X has the following distribution (X^n iid $\mathcal{N}(0, P/n_t)$)

$$X^n \frac{\sqrt{nTP}}{\|X^n\|_F}$$

And Q_Y is the output distribution when the input is iid Gaussian $\mathcal{N}(0, P/n_t)$. Both of these are fairly easily adjustable in the code.

Note that the `iter` variable in the code sets the number of Monte Carlo iterations. The sampling loop uses `parfor`, which attempts to distribute the computation. The number of Monte Carlo iterations should be taken $< 100/\epsilon$. For small ϵ (e.g. 10^{-3}, you may want to let the code run overnight.

6.3 Normal Approximation: `MIMO_csir_normapx()`

Format:

```
function logM = MIMO_csir_normapx(n,epsilon,n_t,n_r,T,P,real_or_complex)
```

This function computes the normal approximation up to the $O(\sqrt{n})$ term, i.e.

$$\log M \approx nTC - \sqrt{nTV}Q^{-1}(\epsilon)$$

It utilizes one of two functions to compute the capacity C and dispersion V:

```
function [C,V] = capacity_and_dispersion(n_t,n_r,T,P)
```

or

```
function [C,V] = capacity_and_dispersion_mc(n_t,n_r,T,P,real_or_complex)
```

The `capacity_and_dispersion` function computes C and V via numerical integration for the complex MIMO channel, while `capacity_and_dispersion_mc` compute these quantities using Monte Carlo. The Monte Carlo computation is also useful it's flexibility, e.g. it's simple to change the fading distribution to compute the capacity and dispersion of non-Rayleigh fading channels.

Chapter 7

Energy-per-bit under vanishing spectral efficiency

General info:

- Maintainer: Y. Polyanskiy <yp@mit.edu>
- Main references: [5, 6]
- Example: See Fig. 7.1
- Channel model:

$$Y_j = H_j X_j + Z_j, \quad j = 1, \dots \qquad n = \infty \quad \textbf{(!!!)} \tag{7.1}$$

  - Noise: $Z_j \sim \mathcal{CN}(0, N_0)$ iid and N_0 is noise spectral density
  - Channel gain (for AWGN): $H_j = 1$ constant
  - Channel gain (for Rayleigh fading): $H_j \sim \mathcal{CN}(0, 1)$ iid
  - Power constraints: Each codeword x^∞ satisfies

$$\sum_{j=1}^{\infty} |x_j|^2 \le E\,,$$

    where E is the total energy budget.
  - Energy-per-bit: For the codebook $\mathcal{C}$ of size M its energy-per-bit is defined as

$$E_b = \frac{E}{\log_2 M}\,.$$

- Description:
  Bounds in this Chapter address the question of finding the absolutely minimal energy-per-bit required for transmitting $\log_2 M$ bits with probability of error ϵ and *without restriction on the number of channel uses*. Asymptotically as $M \to \infty$ it is known:

$$\frac{E_b}{N_0} \to \log_e 2 \approx -1.59\ dB$$

```
> energy-per-bit/plot_all()
```

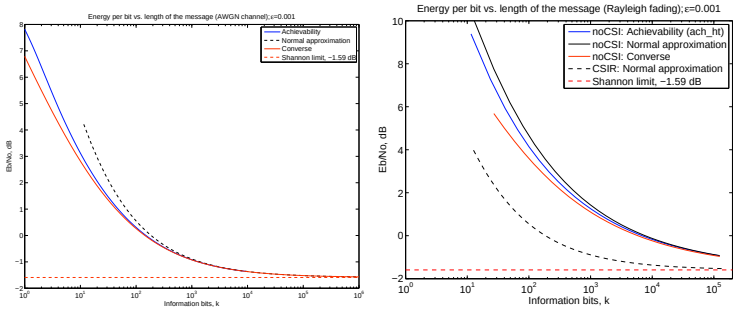

Figure 7.1: Example figures generated by the energy-per-bit code

- Real/Complex: In the regime of infinite bandwidth as here, there is no difference between real/complex channel inputs (but noise variance in the real case should be taken $Z_j \sim \mathcal{N}(0, \frac{N_0}{2})$).
- MIMO and block-fading: Surprisingly, having multiple transmit antennas or block-constant fading makes no difference for energy-per-bit, see [6]. Having $m_r > 1$ receive antennas is equivalent to scaling $N_0 \to \frac{N_0}{m_r}$ and thus again does not require any special treatment.
- Remark: These bounds correspond to allowing no feedback link. Even a modicum of feedback significantly improves the energy-per-bit tradeoff at finite length, see [5].
- Remark: The tradeoff $\frac{E_b}{N_0}$ vs. $\log_2 M$ can also be interpreted as the rate-blocklength tradeoff for the power-constrained *infinite-bandwidth* continunuous-time channel, see [5, (50)-(51)].
- Common input/output arguments:
  1. `epsil` – block probability of block error.
  2. `lm` or `Lms` – input or output argument; $\log_2 M$, log-size of codebook (base-2 information units).
  3. `en0` or `EE` – input/output argument; total energy budget available `EE`$= \frac{E}{N_0}$ (to get $\frac{E_b}{N_0}$ divide by `lm`).

7.1 Summary plot

```
plot_all()
```

This script generates two figures, shown on Fig. 7.1.
Note: Script takes several hours to finish.

7.2 The AWGN channel ($H_j = 1$)

Here $H_j = 1$ in the channel model (7.1).
Format:

```
function En0 = energy_awgn_ach(Lms, epsil);
function lm = energy_awgn_conv(en0, epsil);
function lm = energy_awgn_normapx(en0, epsil);
```

These functions return achievability/converse/normal approximation. The bounds are from [5, Theorems 2 and 3].

7.3 Rayleigh fading: CSIR case

Format:

```
function lm = energy_csir_conv(EE, epsil);
```

This function computes the converse (upper) bound on the number of information bits, `lm`, that a Rayleigh fading channel with perfect CSIR can transmit under a total energy budget `EE`$= \frac{E}{N_0}$; see [6, Theorem 11]. Note that for CSIR case we can use `energy_aegn_ach()` and `energy_awgn_normapx()` for achievability and normal approximation, see [6, Section III.D].

Note: vector `EE` should be in ascending order.

7.4 Rayleigh fading: noCSI case

Format:

```
function lm = energy_nocsi_ach_ht(EE, epsil);
function lm = energy_nocsi_ach_ml(EE, epsil);
```

These are two achievability bounds [6, Corollary 4 and 5] with the second one, `_ml()`, being typically better.

```
function lm = energy_nocsi_conv(EE, epsil);
function lm = energy_nocsi_normapx(EE, epsil);
```

The converse bound is [6, Theorem 6]. The normal approximation is computed by maximizing over A in [6, (64)].

Note: Vector `EE` should be in ascending order.
Note: Functions `energy_nocsi_ach_ml()` and `energy_nocsi_conv()` are *very slow*.

Chapter 8

Fixed-length source coding under a distortion criterion

- Maintainer: V. Kostina <vkostina@caltech.edu>
- Main references: [7]
- Performance criterion - excess distortion probability:

$$\mathbb{P}[\mathsf{d}(S, Z) > d] \leq \epsilon,$$

  where S is the source and Z is its representation.
- The code uses auxiliary functions contained in `lib/`. This directory can to be added to Matlab path by running the following command: `path(path, '../../lib');`
- To speed up calculations, some functions use the so-called 'persistent' variables whose values are retained during multiple calls to those functions. This helps to choose the starting positions for various optimizations wisely. For example, if the function is computed for blocklength 510 after blocklength 500, it takes advantage of the previous computation to choose a starting position for various optimizations. To avoid problems, use command `clear all` to clear the values of all persistent variables when making a computation unrelated to the previous computation.
- Recommended usage:

```
path(path, '../../lib');
clear all;

N = 10:10:1000;
out = NaN(1, length(N));
for k = 1:length(N)
    n = N(k);
    try
        out(k) = Rstar(..., n, ...);
        save('mydatafile');
    catch ME
```

```
> cd sc/GMS
> plot_all(0.1, 0.01, 10:5:500)
```

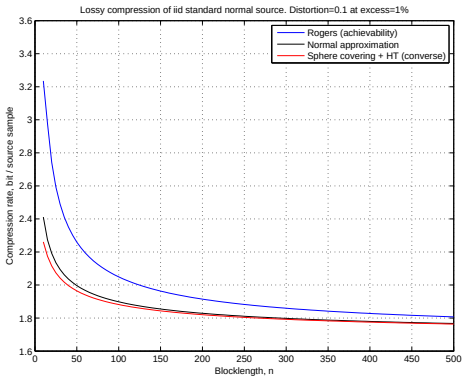

Figure 8.1: Lossy compression bounds for the memoryless Gaussian source.

```
            fprintf('Error: %s\n', getReport(ME));
            out(k) = NaN;
        end
    end
```

- For an example plot (produced via `sc/GMS/plot_all.m`) see Fig. 8.1.

8.1 Binary memoryless source

Source model

$S_1, S_2, \ldots$ are i.i.d Bernoulli with bias p.

Distortion measure

Bit error rate

$$\mathsf{d}(s^k, z^k) = \frac{1}{k} \sum_{i=1}^{k} 1\{s_i \neq z_i\}.$$

Matlab code

`sc/BMS/`

Main functions

`Rstar(d, n, e, p, fun)`: Compression rate compatible with a given excess distortion.

`Dstar(R, n, e, p, fun)`: Distortion threshold compatible with a given excess probability and a given rate.

Input arguments

`d:` distortion threshold.

`R:` compression rate.

`n:` blocklength.

`e:` excess distortion probability.

`p:` source bias.

`fun:` alias for the function to be computed. Acceptable values:

- *'shannon'*: Shannon's achievability bound [7, Theorem 1].
- *'normal'*: Gaussian approximation [7, Theorem 12].
- *'spherecoveringc'*: Converse via sphere covering and hypothesis testing [7, Theorem 8].
- *'generalc'*: General converse via `d`-tilted information [7, Theorem 7].
- *'spherecoveringa'*: Achievability via covering with random spheres [7, Theorem 21].
- *'permutationa'*: Achievability via permutation coding (i.e. all codewords are of the same type, namely, the type that achieves the rate-distortion function) [7, Theorem 22].
- *'martona'*: Converse via Marton's bound [7, Theorem 5].

8.2 Gaussian memoryless source

Source model

$S_1, S_2, \ldots$ are i.i.d $\mathcal{N}(0,1)$.

Distortion measure

Squared error

$$\mathsf{d}(s^k, z^k) = \frac{1}{k}\sum_{i=1}^{k}(s_i - z_i)^2$$

Matlab code

`sc/GMS/`

Main functions

`Rstar(d, n, e, fun):` Compression rate compatible with a given excess distortion.

`Dstar(R, n, e, fun):` Distortion threshold compatible with a given excess probability and a given rate.

Input arguments

`d:` distortion threshold.

`R:` compression rate.

`n:` blocklength.

`e:` excess distortion probability.

`fun:` alias for the function to be computed. Acceptable values:

- *'shannon'*: Shannon's achievability bound [7, Theorem 1].
- *'normal'*: Gaussian approximation [7, Theorem 12].
- *'spherecoveringc'*: Converse via sphere covering and hypothesis testing [7, Theorem 8].
- *'spherecoveringa'*: Achievability via covering with random spheres [7, Theorem 37].

The following `fun` values are accepted only by `Rstar(...)`.

- *'generalc'*: General converse via `d`-tilted information [7, Theorem 7].
- *'rogersa'*: Achievability using a result by Rogers [7, Theorem 39].

8.3 Binary erased source

Source model

$S_1, S_2, \ldots$ are i.i.d fair coin flips. However, the encoder does not observe $S_1, S_2, \ldots$. Instead, it observes an erased version of the source: $X_1, X_2, \ldots$, where $X_i = S_i$ with probability $1 - \alpha$, and $X_i = \texttt{?}$ with probability α.

Distortion measure

Bit error rate with respect to the original source

$$\mathtt{d}(s^k, z^k) = \frac{1}{k} \sum_{i=1}^{k} 1\{s_i \neq z_i\}, \qquad s_i, z_i \in \{\texttt{0}, \texttt{1}\}.$$

Matlab code

`sc/BES/`

Main functions

`Rstar(d, n, e, alpha, fun):` Compression rate compatible with a given excess distortion.

Input arguments

`d:` distortion threshold.

`n:` blocklength.

`e:` excess distortion probability.

`alpha:` erasure probability.

`fun:` alias for the function to be computed. Acceptable values:

- *'spherecoveringa'*: Achievability via covering with random spheres [7, Theorem 33].
- *'spherecoveringc'*: Converse via sphere covering (hypothesis testing) [7, Theorem 32].
- *'normal'*: Gaussian approximation [7, Theorem 12].
- *'aeq'*: Achievability via covering with random spheres for a surrogate rate-distortion problem which negates the stochastic variability of the erasure channel [8].
- *'ceq'*: Converse via sphere covering (hypothesis testing) for the surrogate problem [8].
- *'normaleq'*: Gaussian approximation for the surrogate problem [8].

Chapter 9

Variable-length compression allowing errors

- Maintainer: V. Kostina <vkostina@caltech.edu>
- Main references: [9].
- Fixed probability of error:

$$\mathbb{P}[S \neq Z] \leq \epsilon,$$

  where S is the source and Z is its reconstruction after the (almost lossless) encoding.
- Performance criterion: average length of the compressed representation.
- Performance criterion - excess distortion probability:

$$\mathbb{P}[\mathsf{d}(S, Z) > d] \leq \epsilon,$$

  where S is the source and Z is its reconstruction after the transmission over a noisy channel.
- The code uses auxiliary functions contained in `lib/`. This directory needs to be added to Matlab path by running the following command: `path(path, '../../lib');`
- To speed up calculations, some functions use the so-called 'persistent' variables whose values are retained during multiple calls to those functions. This helps to choose the starting positions for various optimizations wisely. For example, if the function is computed for blocklength 510 after blocklength 500, it takes advantage of the previous computation to choose a starting position for various optimizations. To avoid problems, use command `clear all` to clear the values of all persistent variables when making a computation unrelated to the previous computation.
- Recommended usage:

```
path(path, '../../lib');
clear all;
```

```
N = 10:10:1000;
out = NaN(1, length(N));
for k = 1:length(N)
    n = N(k);
    try
        out(k) = Rstar(..., n, ...);
        save('mydatafile');
    catch ME
        fprintf('Error: %s\n', getReport(ME));
        out(k) = NaN;
    end
end
```

9.1 Binary memoryless source

Source model

$S_1, S_2, \ldots$ are i.i.d Bernoulli with bias p.

Matlab code

`sc/varrate/`

Main function

`Rstar(p, e, n, fun)`: Compression rate compatible with a given excess distortion.

Input arguments

`p`: source bias.

`e`: error probability.

`n`: blocklength.

`fun`: alias for the function to be computed. Acceptable values:

'approx': Gaussian approximation [9, Theorem 4].

'exact': Exact minimum average rate [9, (26)].

'exacti': Exact value of the Erokhin function [9, (40)].

'ie': Exact value of the expectation of the ϵ-cutoff of information in S^k [9, (11),(13)].

Chapter 10

Joint source-channel coding under a distortion criterion

- Maintainer: V. Kostina <vkostina@caltech.edu>
- Main references: [10]
- Performance criterion - excess distortion probability:

$$\mathbb{P}[\mathsf{d}(S, Z) > d] \leq \epsilon,$$

  where S is the source and Z is its reconstruction after the transmission over a noisy channel.

- The code uses auxiliary functions contained in `lib/`. This directory needs to be added to Matlab path by running the following command: `path(path, '../../lib');`
- To speed up calculations, some functions use the so-called 'persistent' variables whose values are retained during multiple calls to those functions. This helps to choose the starting positions for various optimizations wisely. For example, if the function is computed for blocklength 510 after blocklength 500, it takes advantage of the previous computation to choose a starting position for various optimizations. To avoid problems, use command `clear all` to clear the values of all persistent variables when making a computation unrelated to the previous computation.
- Recommended usage:

```
path(path, '../../lib');
clear all;

N = 10:10:1000;
out = NaN(1, length(N));
for k = 1:length(N)
    n = N(k);
    try
        out(k) = Rstar(..., n, ...);
        save('mydatafile');
```

```
        catch ME
            fprintf('Error: %s\n', getReport(ME));
            out(k) = NaN;
        end
    end
```

10.1 Binary memoryless source transmitted over a binary symmetric channel

Source model

$S_1, S_2, \ldots$ are i.i.d Bernoulli with bias p.

Channel model

$Y_i = X_i + N_i$, where N_i are i.i.d. Bernoulli with bias δ.

Distortion measure

Bit error rate

$$\mathsf{d}(s^k, z^k) = \frac{1}{k} \sum_{i=1}^{k} 1\{s_i \neq z_i\}.$$

Matlab code

`jscc/BMS-BSC/`

Main functions

`Rstar(p, delta, d, e, n, fun)`: Compression rate compatible with a given excess distortion.

`Dstar(R, n, e, p, delta, fun)`: Distortion threshold compatible with a given excess probability and a given rate.

Input arguments

`p`: source bias.

`delta`: channel crossover probability.

`d`: distortion threshold.

`e`: excess distortion probability.

`R`: compression rate.

`n`: blocklength.

`fun`: alias for the function to be computed. Acceptable values:

'approx': Gaussian approximation [10, Theorem 10].

'clistdecoding': Converse via list decoding and hypothesis testing [10, Theorem 4].

'ajoint': Achievability via joint source-channel coding [10, Theorem 8].

'aseparate': Achievability via separation [10, Theorem 6].

The following `fun` values are accepted only by `Rstar(...)`.

'cgeneral': General converse via `d`-tilted information [10, Theorem 2].

'approxseparate': Gaussian approximation for the separation architecture [10, (126)].

'arcu': For the almost lossless transmission, achievability via RCU bound [10, Theorem 9].

The following `fun` values are accepted only by `Dstar(...)`.

'random': Achievability via random coding for fair coin flip source [10, Theorem 6].

'nocoding': Achievability via no coding. [10, Theorem 23].

10.2 Gaussian source transmitted over an AWGN channel

Source model

$S_1, S_2, \ldots$ are i.i.d $\mathcal{N}(0,1)$.

Channel model

$Y_i = X_i + N_i$, where N_i are i.i.d. $\mathcal{N}(0,1)$.

Distortion measure

Squared error

$$\mathsf{d}(s^k, z^k) = \frac{1}{k}\sum_{i=1}^{k}(s_i - z_i)^2.$$

Matlab code

`jscc/GMS-AWGN/`

Main functions

`Rstar(P, d, e, n, fun)`: Transmission rate compatible with a given excess distortion (nats).

`Dstar(R, n, e, P, fun)`: Distortion threshold compatible with a given excess probability and a given rate.

Input arguments

`P:` maximum allowable transmit power, or SNR.

`d:` distortion threshold.

`e:` excess distortion probability.

`R:` transmission rate.

`n:` blocklength.

`fun:` alias for the function to be computed. Acceptable values:

'approx': Gaussian approximation [10, Theorem 10].

'cht': Converse via list decoding and hypothesis testing [10, Theorem 4].

'ajoint': Achievability via joint source-channel coding [10, Theorem 8].

'aseparate': Achievability via separation [10, Theorem 6].

The following `fun` values are accepted only by `Dstar()`.

'nocoding': Achievability via no coding [10, Theorem 24].

'approxnocoding': Gaussian approximation for the uncoded system [10, (129), (208)].

The following `fun` values are accepted only by `Rstar()`.

'cgeneral': General converse via `d`-tilted information [10, Theorem 2].

'approxseparate': Gaussian approximation for the separation architecture [10, (126)].

Chapter 11

The binary-symmetric channel (BSC)

General info:

- Maintainer: Y. Polyanskiy <yp@mit.edu>
- Main references: [1]
- Example: See Fig. 11.1
- Channel model:

$$Y_j = X_j + Z_j \mod 2, \quad j = 1, \ldots, n$$

  - Codewords are 0/1: $x_j \in \{0, 1\}, j = 1, \ldots, n$
  - Noise: $Z_j \sim \mathrm{Ber}(\delta)$ iid.

- Common input/output arguments:
  1. `delta` – input argument; crossover probability δ.
  2. `epsil` – block probability of block error. *Warning:* All code in this section is tailored to maximal probability of error.
  3. `lm` or `Lms` – output argument; $\log_2 M$, log-size of codebook (base-2 information units).
  4. `n` – input argument; blocklength.
     For slow functions that do not support vectorized arguments.
  5. `Ns` – input argument; vector of blocklengths.

11.1 Summary plot: `plot_all()`

Definition:

```
function [Ns ach_rcu ach_dt ach_gal conv normapx_val] = plot_all(delta, epsil);
```

This function computes and plots various bounds on the same figure. See Fig. 11.1.

```
> plot_all()
```

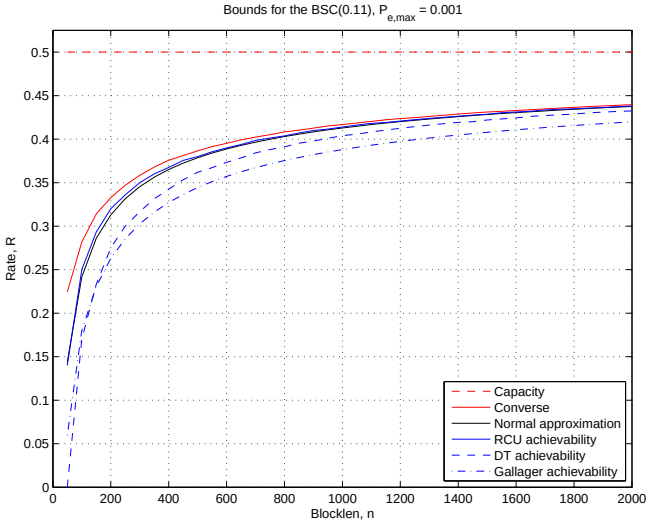

Figure 11.1: Code and resulting picture for BSC bounds

11.2 Achievability: `dt_ach()`

Definition:

```
function Lms = dt_ach(Ns, delta, epsil)
```

Function computes DT achievability bound, see [1, (166)].

11.3 Achievability: `rcu_ach()`

Format:

```
function lm = rcu_ach(n, delta, epsil)
```

This computes the RCU achievability bound, see [1, (162)].
Warning: This function only takes scalar value of n.

11.4 Achievability: `gallager_ach()`

Format:

```
function Lms = gallager_ach(Ns, delta, epsil)
```

Computes Gallager's achievability bound, see [1, (44)].

11.5 Converse: `converse()`

Format:

```
function Lms = converse(Ns, delta, epsil)
```

This computes the sphere-packing lower bound. Equivalently, this is the best possible meta-converse bound with $Q_{Y^n} = \mathrm{Ber}(1/2)^n$, see [1, Theorem 35].

11.6 Normal approximation: `normapx()`

Format:

```
function Lms = normapx(Ns, delta, epsil);
```

Fast and frequently very precise approximation to both achievability and converse:

$$\log M \approx nC - \sqrt{nV}Q^{-1}(\epsilon) + \frac{1}{2}\log n$$

where $C = \log 2 - \delta \log \frac{1}{\delta} - (1-\delta)\log \frac{1}{1-\delta}$ and $V = \delta(1-\delta)\log^2 \frac{1-\delta}{\delta}$.

Chapter 12

The binary-erasure channel (BEC)

General info:

- Maintainer: Y. Polyanskiy <yp@mit.edu>
- Main references: [1]
- Example: See Fig. 11.1
- Channel model:

$$Y_j = \begin{cases} X_j, & \text{with prob.} 1-\delta \\ e, & \text{with prob.} \delta \end{cases} \qquad j = 1, \ldots, n$$

  Codewords are 0/1: $x_j \in \{0, 1\}, j = 1, \ldots, n$

- Common input/output arguments:
  1. `delta` – input argument; crossover probability δ.
  2. `epsil` – block probability of block error. *Warning:* All code in this section is tailored to maximal probability of error.
  3. `lm` or `Lms` – output argument; $\log_2 M$, log-size of codebook (base-2 information units).
  4. `n` – input argument; blocklength.
     For slow functions that do not support vectorized arguments.
  5. `Ns` – input argument; vector of blocklengths.

12.1 Summary plot: `plot_all()`

Definition:

```
function [Ns ach_dt ach_gal conv normapx_val] = plot_all(delta, epsil);
```

This function computes and plots various bounds on the same figure. See Fig. 12.1.

```
> plot_all()
```

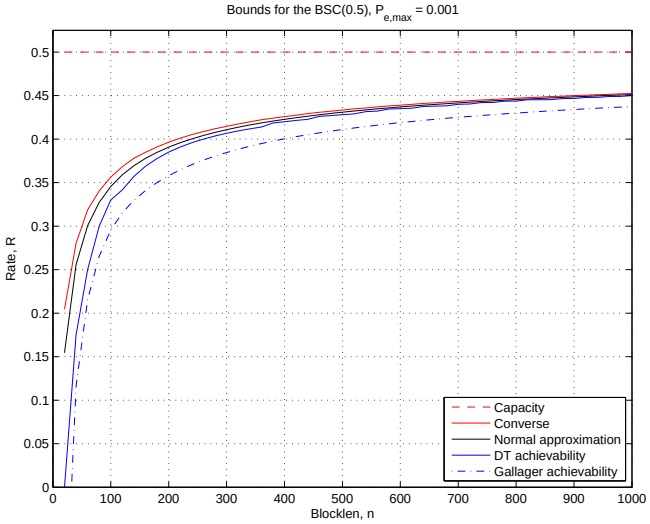

Figure 12.1: Code and resulting picture for BEC bounds

12.2 Achievability: `dt_ach()`

Definition:

```
function Lms = dt_ach(Ns, delta, epsil)
```

Function computes DT achievability bound, see [1, (183)].

Note: The RCU bound is weaker for the BEC, so we do not need to evaluate it.

12.3 Achievability: `gallager_ach()`

Format:

```
function Lms = gallager_ach(Ns, delta, epsil)
```

Computes Gallager's achievability bound, see [1, (44)].

12.4 Converse: `converse()`

Format:

```
function Lms = converse(Ns, delta, epsil)
```

This computes the special converse bound for the BEC, see [1, Theorem 38]. Later, it was found that it corresponds to the meta-converse bound with a non-product distribution Q_{Y^n}, see [11, Section VI.D].

12.5 Normal approximation: normapx()

Format:

```
function Lms = normapx(Ns, delta, epsil);
```

Fast and frequently very precise approximation to both achievability and converse:

$$\log M \approx nC - \sqrt{nV}Q^{-1}(\epsilon)\,,$$

where $C = (1-\delta)\log 2$ and $V = \delta(1-\delta)\log^2 2$.

Chapter 13

The binary-AWGN channel (BI-AWGN)

General info:

- Maintainer: T. Erseghe <erseghe@dei.unipd.it>
- Main references: [12]
- Example: See Fig. 13.1
- Channel model:

$$Y_j = \sqrt{\Omega}\, X_j + Z_j, \quad j = 1, \ldots, n$$

  - Codewords are 0/1: $x_j \in \{\pm 1\}, j = 1, \ldots, n$
  - Noise: $Z_j \sim \mathcal{N}(0, 1)$ iid
  - Ω is the signal to noise ratio (SNR).

- Common input/output arguments:
  1. `n` – blocklength.
  2. `Pe` – block probability of block error, or packet error rate. *Warning:* All code in this section is tailored to average probability of error.
  3. `R` – code rate.
  4. `SNRdB` – signal to noise ratio in dB.

13.1 Converse: `converse_mc()`

Definition #1:

```
R = converse_mc(N,Pe,SNRdB,'approx')
```

returns the metaconverse Polyanskiy-Poor-Verdù (PPV) upper bound on rate for a codeword of length `N`, a packet error rate of `Pe`, and a signal to noise ratio of `SNRdB` (in dB). Multiple bound values can be obtained by ensuring that `N`, `Pe`, and `SNRdB` have the same length. The specific approximation used for calculating the bound is specified by `'approx'`:

- `'normal'` normal approximation, the $O(n^{-1})$ approximation available from [12, Fig. 6, 3rd block]; this is a very fast algorithm, but may provide a weak approximation.
- `'On2'` (default) the $O(n^{-2})$ approximation derived from [12, Fig. 6, 2nd block]; at low `Pe` this is much more reliable than the `'normal'` option.
- `'On3'` the $O(n^{-3})$ approximation derived from [12, Fig. 6, 2nd block]; at high `Pe` this is much more unrealiable than `'On2'`. It can be used to identify the region where `'On2'` is a reliable solution, namely the region where `'On2'` and `'On3'` closely match.
- `'full'` (this option will be available soon) the full bound derived numerically from [12, Fig. 6, 1st block].

Definition #2:

```
SNRdB = converse_mc(N,Pe,R,'approx','error')
```

for a codeword of length `N`, and code rate `R`, the function returns the signal to noise ration `SNRdB` (in dB) at which the PPV metaconverse lower bound on error rate is equal to `Pe`. Multiple bound values can be obtained by ensuring that `N`, `Pe`, and `R` have the same length. The specific approximation used for calculating the bound is specified by `'approx'` (see above).

13.2 Example 1: `example_speff.m`

In `example_speff.m` we use the `converse_mc` function in order to derive an upper bound on rate, R, and then plot a bound on spectral efficiency, namely $\rho = 2R$. We also recall the relation

$$\Omega = \rho \frac{E_b}{N_0}$$

with Ω the SNR, and E_b the energy spent per bit of information. The example is built on a codeword of length `n=1000` for which the bounds at `Pe=1e-5` are derived according to the following code:

```
n = 1e3;
Pe = 1e-5;
SNRdB = -15:10; % SNR Omega
uno = ones(size(SNRdB)); % vector of ones

% normal approximation - expected execution time 2 sec
rhoNA = 2*converse_mc(n*uno,Pe*uno,SNRdB,'normal');
EbN0NA = SNRdB-10*log10(rhoNA);

% O(n^-2) approximation - expected execution time 90 sec
rhoPPV = 2*converse_mc(n*uno,Pe*uno,SNRdB,'On2');
EbN0PPV = SNRdB-10*log10(rhoPPV);
```

```
% O(n^-3) approximation - expected execution time 120 sec
rhoPPVb = 2*converse_mc(n*uno,Pe*uno,SNRdB,'On3');
EbN0PPVb = SNRdB-10*log10(rhoPPVb);
```

and then the results is plotted by using

```
figure
set(0,'defaulttextinterpreter','latex')
semilogy(EbN0PPV,rhoPPV,'-',EbN0PPVb,rhoPPVb,'--',...
            EbN0NA,rhoNA,'-.')
xlabel('SNR $E_b/N_0$')
ylabel('spectral efficiency $\rho$ [bit/s/Hz]')
grid
```

The result is given in the plot on the left in Fig. 13.1

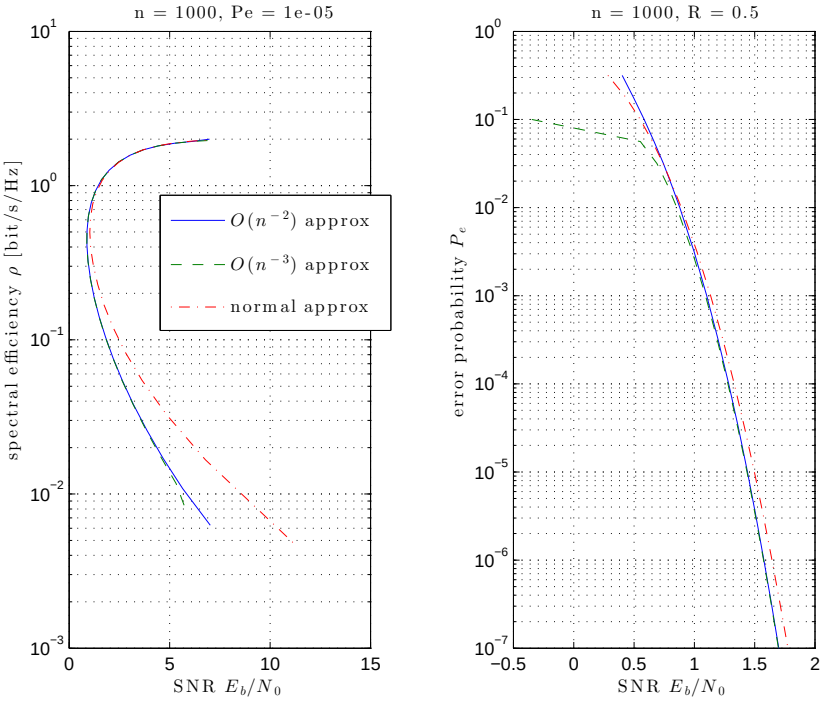

Figure 13.1: Examples of BI-AWGN metaconverse bounds

13.3 Example 2: `example_per.m`

In `example_per.m` we use the `converse_mc` function in order to derive a lower bound on the packet error rate, P_e. The example is built on a codeword of length `n=1000` and rate `R=0.5` for which bounds are derived according to the following code:

```
n = 1e3;
R = 1/2;
Pev = 10.^([-0.5:-0.25:-1.75,-2:-0.5:-7]);
uno = ones(size(Pev));

% normal approximation - expected execution time 15 sec
SNRdBNA = converse_mc(n*uno,Pev,R*uno,'normal','error');

% O(n^-2) approximation - expected execution time 120 sec
SNRdBPPV = converse_mc(n*uno,Pev,R*uno,'On2','error');

% O(n^-3) approximation - expected execution time 280 sec
SNRdBPPVb = converse_mc(n*uno,Pev,R*uno,'On3','error');
```

and then the results is plotted by using

```
figure
set(0,'defaulttextinterpreter','latex')
semilogy(SNRdBPPV,Pev,'-',SNRdBPPVb,Pev,'--',...
          SNRdBNA,Pev,'-.')
xlabel('SNR $E_b/N_0$')
ylabel('error probability $P_e$')
grid
```

The result is given in the plot on the right in Fig. 13.1

Bibliography

[1] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.

[2] G. Durisi, T. Koch, J. Östman, Y. Polyanskiy, and W. Yang, "Short-packet communications over multiple-antenna Rayleigh-fading channels," Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.7512

[3] W. Yang, G. Durisi, T. Koch, and Y. Polyanskiy, "Quasi-static multiple-antenna fading channels at finite blocklength," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 4232–4265, Jul. 2014.

[4] W. Yang, G. Caire, G. Durisi, and Y. Polyanskiy, "Optimum power control at finite blocklength," *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4598–4615, Sep. 2015.

[5] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Minimum energy to send k bits with and without feedback," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 4880–4902, Aug. 2011.

[6] W. Yang, G. Durisi, and Y. Polyanskiy, "Minimum energy to send k bits over multiple-antenna fading channels," *arXiv:1507.03843v1*, Jul. 2015.

[7] V. Kostina and S. Verdú, "Fixed-length lossy compression in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 3309–3338, June 2012.

[8] ——, "Nonasymptotic noisy lossy source coding," in *Proceedings 2013 IEEE Information Theory Workshop*, Seville, Spain, Sep. 2013.

[9] V. Kostina, Y. Polyanskiy, and S. Verdú, "Variable-length compression allowing errors," *IEEE Transactions on Information Theory*, vol. 61, no. 9, pp. 4316–4330, Aug. 2015.

[10] V. Kostina and S. Verdú, "Lossy joint source-channel coding in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2545–2575, May 2013.

[11] Y. Polyanskiy, "Saddle point in the minimax converse for channel coding," *IEEE Trans. Inf. Theory*, May 2012, submitted. [Online]. Available: http://people.lids.mit.edu/yp/homepage

[12] T. Erseghe, "Coding in the finite-blocklength regime: Bounds based on laplace integrals and their asymptotic approximations," *arXiv preprint arXiv:1511.04629*, 2015.