Downlink MIMO Vector Perturbation Precoding to QUBO

Karthik Mudakalli

MIMO basics

- MIMO stands for Multiple Input Multiple Output
- Inputs are transmit antennas at the base station
- Outputs are receive antennas at user devices
- Let N_t be the number of transmit antennas
- Let N_r be the number of users
- The channel is a complex matrix $H \in \mathbb{C}^{N_r \times N_t}$
- Entry h_{ij} is a complex number describing how the signal from transmit antenna j arrives at user i with amplitude scaling and phase shift
- If the base station sends a complex vector $x \in \mathbb{C}^{N_t}$ then the users receive $y \in \mathbb{C}^{N_r}$
- The linear model is $y = Hx + n$ where n is additive noise
- Real and imaginary parts of n are commonly modeled as independent zero mean Gaussian variables with equal variance
- This is called complex additive white Gaussian noise

Why complex entries?

- Wireless signals have both amplitude and phase
- A complex number $ae^{j\theta} = a\cos(\theta) + ja\sin(\theta)$ captures both in one value
- Channel coefficients h_{ij} are complex since they describe amplitude scaling and phase shift from antenna j to user i
- Transmit vector $x \in \mathbb{C}^{N_t}$ has complex entries because antennas send complex-valued modulation symbols (e.g. QPSK, QAM)
- Receive vector $y \in \mathbb{C}^{N_r}$ is also complex because received signals still carry amplitude and phase
- Restricting to real numbers would lose phase information, which is essential in wireless communication

Downlink and interference

Downlink means the base station sends signals to users. The challenge is that one transmit vector must carry the data for all users at once. This creates multiuser interference since each user receives a mixture of all streams unless we pre process the signal. We handle this by precoding at the transmitter.

Symbols and constellations

Each user wants to receive a complex symbol that represents bits. The set of allowed symbols is a constellation. A common constellation is QPSK which stands for Quadrature Phase Shift Keying. The QPSK set is $\{1+j, 1-j, -1+j, -1-j\}$. These four points lie at plus or minus one on the real axis and plus or minus one on the imaginary axis. Since there are four symbols each symbol carries two bits. The vector of user symbols is $u \in \mathbb{C}^{N_r}$ where the ith entry is the symbol for user i.

QPSK in more words

QPSK stands for Quadrature Phase Shift Keying. It uses four points on the complex plane at coordinates $\pm 1 \pm j$. Each point can be labeled by two bits. For example one mapping is 00 to $1 + j$, 01 to $1 - j$, 10 to $-1 + j$, and 11 to $-1 - j$. The mapping is fixed by the system design. The key property is constant energy and equal distance between symbols along both axes.

What is precoding and zero forcing

Without processing if we transmit u each user receives a mixture of all symbols weighted by channel coefficients. Precoding shapes the transmit vector x so that users do not interfere. We focus on zero forcing precoding. The zero forcing precoder is

$$P = H^H \left(HH^H\right)^{-1}$$

when HH^H is invertible. The goal is that HP is close to the identity so that if we send $x = Pd$ then $y = HPd \approx d$. Each user receives mostly its own symbol.

Why vector perturbation

Zero forcing can result in large transmit power when H is poorly conditioned. Vector perturbation precoding reduces power by adding an integer shift before precoding. The transmit vector is

$$x = P\,(u + \tau v)$$

where $v \in \mathbb{Z}[j]^{N_r}$ is the perturbation vector
In practice, each real and imaginary part satisfies $\Re(v_i), \Im(v_i) \in \{-1, 0, 1\}$.

ZF or MMSE in Vector Perturbation Precoding

Vector perturbation precoding builds on a linear precoder that cancels interference. Two standard options exist:

$$\text{ZF: } P = H^H(HH^H)^{-1}, \qquad \text{MMSE: } P = H^H(HH^H + \alpha I)^{-1}.$$

- ZF (zero forcing) completely removes interference but can amplify noise when H is ill-conditioned.
- MMSE (minimum mean square error) trades off interference suppression and noise amplification using $\alpha > 0$.
- Vector perturbation adds an integer offset before applying P:

  $$x = P(u + \tau v),$$

  where $v \in \mathbb{Z}[j]^{N_r}$.
- The goal is to choose v that minimizes the transmit power while preserving symbol detectability after modulo reduction at the receivers.

VPP objective

The base station chooses v to minimize transmit energy

$$v^\star = \arg\min_{v \in \mathbb{Z}^{N_r}} \ \left\| P \left(u + \tau v \right) \right\|_2^2 .$$

This is a closest vector problem in the lattice generated by the columns of P. Our goal is to reformulate this integer optimization as a QUBO so that it can be handled by solvers that accept quadratic objectives over binary variables.

Why convert to real

Optimization over binary variables is most convenient in the real domain. We map complex vectors and matrices to real ones. If $z = a + jb \in \mathbb{C}^n$ define

$$\phi(z) = \begin{bmatrix} a \\ b \end{bmatrix} \in \mathbb{R}^{2n}.$$

If $P = A + jB \in \mathbb{C}^{m \times n}$ define

$$\Phi(P) = \begin{bmatrix} A & -B \\ B & A \end{bmatrix} \in \mathbb{R}^{2m \times 2n}.$$

This mapping preserves matrix vector multiplication in the sense that $\phi(Pz) = \Phi(P)\phi(z)$ and preserves norms so that $\|Pz\|_2^2 = \|\Phi(P)\phi(z)\|_2^2$.

After realification, the perturbation vector v becomes real-valued with dimension $2N_r$, containing both real and imaginary parts explicitly.

Define F, y, and G

Let $F = \Phi(P) \in \mathbb{R}^{2N_t \times 2N_r}$ and let $y = \phi(u) \in \mathbb{R}^{2N_r}$ denote the real-valued representations of the complex precoder and user symbol vector, respectively. After stacking real and imaginary parts, the perturbation vector becomes a real integer vector $v \in \mathbb{Z}^{2N_r}$ derived from the complex Gaussian integer vector $v \in \mathbb{Z}[j]^{N_r}$.
The transmit energy becomes

$$\|P\,(u + \tau v)\|_2^2 = \|F\,(y + \tau v)\|_2^2 .$$

Define the Gram matrix $G = F^T F \in \mathbb{R}^{2N_r \times 2N_r}$. The energy is

$$E(v) = (y + \tau v)^T \, G \, (y + \tau v) ,$$

which is a real quadratic function of the integer vector v.

Range of the Perturbation Vector v

Originally, $v \in \mathbb{Z}[j]^{N_r}$ could include any Gaussian integer values. However, in practical implementations and as confirmed by the referenced author, the perturbation components are small.
Clarification: With high probability,

$$\Re(v_i), \Im(v_i) \in \{-1, 0, 1\}.$$

This means that each complex component of v can take one of nine possible values:

$$v_i \in \{-1, 0, 1\} + j\{-1, 0, 1\}.$$

After realification, v becomes a real integer vector

$$v_{\text{realified}} = \begin{bmatrix} \Re(v) \\ \Im(v) \end{bmatrix} \in \mathbb{Z}^{2N_r},$$

where each entry is restricted to $\{-1, 0, 1\}$. This bound is sufficient for accurate performance in practice and simplifies brute force checks.

What is a QUBO

A QUBO is a quadratic unconstrained binary optimization. The unknown is a binary vector $q \in \{0, 1\}^M$. The objective is a quadratic form

$$E(q) = q^T Qq + \text{offset}$$

where Q is a real symmetric matrix. Our task is to represent the integer vector v with a binary vector q so that the original energy becomes a quadratic function of q.

Encode integers with binary variables

Each integer component of the realified perturbation vector is encoded in binary. We use two's complement with t magnitude bits so each integer v_i uses $t+1$ bits. The weights are $[-2^t, 2^0, 2^1, \dots, 2^{t-1}]$. For example, if $t = 2$, then the weights are $[-4, 1, 2]$ and the represented integers range from -4 to 3.

The parameter t controls the **bit depth and dynamic range** of each integer variable: increasing t expands the search space for the perturbation values.

We build a block-diagonal encoding matrix $C \in \mathbb{R}^{(2N_r)\times M}$ with one weight block per integer so that

$$v = Cq, \qquad q \in \{0,1\}^M, \qquad M = (t+1)\cdot 2N_r.$$

We also keep a decode function $q \mapsto v = Cq$ to interpret solutions.

Derive the QUBO

Start from the real energy

$$E(v) = (y + \tau v)^T \, G \, (y + \tau v) \, .$$

Substitute $v = Cq$ to get

$$E(q) = (y + \tau Cq)^T \, G \, (y + \tau Cq) \, .$$

Use $(a+b)^T G(a+b) = a^T Ga + 2a^T Gb + b^T Gb$ with $a = y$ and $b = \tau Cq$:

$$E(q) = y^T Gy + 2\tau \, y^T GC \, q + \tau^2 \, q^T C^T GC \, q.$$

Define $A = \tau^2 C^T GC$ and $b^T = 2\tau \, y^T GC$. Make A symmetric by $Q \leftarrow \frac{1}{2}(A + A^T)$. Since $q_i^2 = q_i$ for binary q_i we fold the linear term into the diagonal by updating $Q_{ii} \leftarrow Q_{ii} + b_i$. The constant offset is $y^T Gy$. The final QUBO is $E(q) = q^T Qq + \text{offset}$.

Algorithm pipeline

Step one. Input H and u. Step two. Compute $P = H^H(HH^H)^{-1}$. Step three. Realify P and u to F and y. Step four. Compute $G = F^T F$. Step five. Choose t and build C. Step six. Compute $A = \tau^2 C^T GC$, compute $b = 2\tau\, C^T Gy$, and the constant $y^T Gy$. Step seven. Symmetrize A to Q and add b to the diagonal. The QUBO instance is (Q, offset) together with the decoder $v = Cq$.

Code files

- `encoding.py` Implements complex to real mapping and integer encoding.
- `vpp.py` Implements the VPP to QUBO pipeline and a verification routine.
- `vpp_qubo_example.py` Creates random toy instances, builds the QUBO, runs brute force checks, and prints the result.

encoding.py

Function complex_to_real maps complex vectors to stacked real and imaginary parts and maps complex matrices to the standard block form with A and B. This preserves matrix vector multiplication and norms. Function build_integer_encoding takes the number of integer variables and t and returns the block diagonal matrix C with weights $[-2^t, 1, 2, \ldots, 2^{t-1}]$. It also returns a decode function that applies C to a binary vector to recover the integer vector.

encoding.py explained

```
import numpy as np  # Import NumPy for arrays and linear algebra

def complex_to_real(M):
    M = np.asarray(M)  # Ensure M is a NumPy array to safely access .real and .imag

    # vector case
    if M.ndim == 1:  # If M is a 1D complex vector
        # complex vector u = a + j b
        # map to real vector: [Re(u); Im(u)]
        a = M.real   # Extract real part
        b = M.imag   # Extract imaginary part
        return np.concatenate([a, b], axis=0).astype(np.float64)
        # Stack real above imaginary parts into one vector of doubles

    # matrix case
    elif M.ndim == 2:  # If M is a 2D complex matrix
        A = M.real     # Real part of the matrix
        B = M.imag     # Imaginary part of the matrix
        top = np.concatenate([A, -B], axis=1)  # Build block [A  -B]
        bot = np.concatenate([B,  A], axis=1)  # Build block [B   A]
        return np.concatenate([top, bot], axis=0).astype(np.float64)
        # Stack top and bottom blocks to form real-valued block matrix

    else:
        raise ValueError("M must be 1D or 2D")
        # Error if input is not a vector or matrix
```

encoding.py explained continued

```
# builds binary encoding matrix C for n integers
def build_integer_encoding(n_vars, t=1):
    cols_per = t + 1  # Bits per integer (t magnitude bits + 1 sign bit)
    C = np.zeros((n_vars, n_vars * cols_per), dtype=int)
    # Allocate encoding matrix of size (n_vars x n_vars*cols_per)

    # 2’s complement weights: [ -2^t, 1, 2, ..., 2^(t-1) ]
    weights = np.array([-(2**t)] + [2**i for i in range(t)], dtype=int)
    # This defines how binary bits map to signed integer values

    # Fill block diagonal where each integer gets its own set of weights
    for k in range(n_vars):
        C[k, k*cols_per:(k+1)*cols_per] = weights

    def decode(q):
        q = np.asarray(q).reshape(-1)  # Flatten binary vector q
        return (C @ q).astype(int)     # Multiply by C to recover integers

    return C, decode  # Return both the encoding matrix and the decoder
```

vpp.py main steps

Function zf_precoder returns $P = H^H(HH^H)^{-1}$. Function build_basis_map returns $F = \Phi(P)$. Function gram_of_map returns $G = F^T F$. Function form_qubo_core takes G, y, τ, and C and builds Q and the constant offset by computing A, b, symmetrizing A, and adding b to the diagonal. Function qubo_from_vpp ties the steps together. It returns Q, offset, C, and the decode function.

vpp.py verification

Function brute_force_check is a verifier for small toy problems. It does two exhaustive searches. The first is a direct integer search over a small set of integer vectors v that are consistent with the chosen t. It computes $E = (y + \tau v)^T G(y + \tau v)$ and tracks the minimum. The second is a binary search over all q of length M where $M = (t + 1) \cdot 2N_r$. It computes $E\_\text{qubo} = q^T Qq + \text{offset}$ and tracks the minimum then decodes the best q into v. It compares the best energies and the best perturbations. It skips if M is too large since 2^M grows very fast. Agreement confirms that the QUBO encodes the original objective.

vpp.py explained line by line

```
import numpy as np                      # Numerical computations
from itertools import product           # Used for brute-force search
from .encoding import complex_to_real  # Helper to convert complex to real form

# Zero-Forcing (ZF) precoder from 2102.12540v1.pdf
# Formula: P = H^H * (H H^H)^(-1)
def zf_precoder(H):
    H = np.asarray(H, dtype=np.complex128)   # Ensure H is complex128 array
    return H.conj().T @ np.linalg.pinv(H @ H.conj().T)

# Gram matrix G = F^T F
# F = real-valued basis matrix from precoder P
# G encodes inner products of basis vectors; needed to make cost quadratic in v
def gram_of_map(F):
    return F.T @ F

# Convert complex precoder P into real matrix representation
# So optimization is expressed over real numbers (not complex)
def build_basis_map(P):
    return complex_to_real(P)
```

vpp.py explained line by line

```
# Integer-to-binary encoding: creates block-diagonal matrix C
# v = C q  where v is integer vector, q is binary vector
def build_integer_encoding(n, t=1):
    bits_per_int = t + 1                # Each integer encoded with t+1 bits (2's complement)
    m = n * bits_per_int                # Total number of binary variables
    weights = np.array([-(2**t)] + [2**i for i in range(t)], dtype=int)
    # Example: t=2 → weights = [-4, 1, 2]
    C = np.zeros((n, m), dtype=int)
    for i in range(n):
        C[i, i*bits_per_int:(i+1)*bits_per_int] = weights
    def decode(q):                      # Helper: recover integer vector from q
        q = np.asarray(q).reshape(-1)
        return (C @ q).astype(int)
    return C, decode
```

vpp.py explained line by line

```
# Build QUBO cost matrix
def form_qubo_core(G, y_vec, tau, C, const_offset=0.0):
    y = y_vec.reshape(-1, 1)                        # Column vector
    A = tau**2 * (C.T @ G @ C)                      # Quadratic term
    b = 2.0 * tau * (y.T @ G @ C).reshape(-1)       # Linear term
    const = float(y.T @ G @ y) + float(const_offset)  # Constant term
    Q = 0.5 * (A + A.T)                             # Symmetrize Q
    for i in range(Q.shape[0]):                     # Add linear term to diagonal
        Q[i, i] += b[i]
    Q = 0.5 * (Q + Q.T)                             # Final symmetrization
    return Q, const                                 # Return QUBO form

# Full QUBO construction pipeline from VPP
def qubo_from_vpp(H, u, tau, t=1):
    H = np.asarray(H, dtype=np.complex128)         # Channel matrix
    u = np.asarray(u, dtype=np.complex128).reshape(-1)   # User symbols
    Nr = H.shape[0]                                # Number of users
    P = zf_precoder(H)                             # ZF precoder
    F = build_basis_map(P)                         # Realify basis
    u_r = complex_to_real(u)                       # Realify symbols
    C, decode = build_integer_encoding(2*Nr, t=t)   # Encoding matrix
    G = gram_of_map(F)                             # Gram matrix
    Q, const = form_qubo_core(G, u_r, tau, C)  # QUBO form
    return {"Q": Q, "offset": const, "C": C, "decode": decode}
```

vpp.py explained line by line

```
# Verification function (brute force)
def brute_force_check(H, u, tau, Q, offset, C, decode, t=1):
    Nr = H.shape[0]
    F = build_basis_map(zf_precoder(H))        # Build basis
    G = gram_of_map(F)
    y = complex_to_real(u)
    vals = list(range(-(2**t), 2**t))          # Candidate integers
    cand = np.array(list(product(*([vals] * (2*Nr)))), dtype=int)
    # Direct method: try every l
    bestE, bestl = np.inf, None
    for l in cand:
        v = y.reshape(-1) + tau * l
        E = float(v @ G @ v)
        if E < bestE:
            bestE, bestl = E, l.copy()
    # Brute-force QUBO: try every binary vector q
    M = Q.shape[0]
    if M > 26:                                 # Hard cutoff (too large)
        return {"ok": False, "reason": "too many variables",
                "bestE_direct": bestE, "best_l_direct": bestl}
    bestQ, bestq = np.inf, None
    for z in range(1 << M):                    # Check all 2^M binary vectors
        q = np.array([(z >> i) & 1 for i in range(M)], float)
        E = float(q @ Q @ q + offset)
        if E < bestQ:
            bestQ, bestq = E, q
    l_from_qubo = decode(bestq)                # Decode binary solution
    return {"ok": True, "E_direct": bestE, "E_qubo": bestQ,
            "l_from_qubo": l_from_qubo, "l_direct": bestl}
```

vpp.py explained line by line

```
# Random example generator
def sample_and_build(seed=0):
    rng = np.random.default_rng(seed)          # Random generator
    Nr = Nt = 2                                # Fixed small dimension
    # Channel matrix H: complex Gaussian i.i.d. entries, unit variance
    H = (rng.normal(size=(Nr, Nt)) + 1j * rng.normal(size=(Nr, Nt))) / np.sqrt(2)
    # QPSK constellation {±1 ± j}
    const = np.array([1+1j, 1-1j, -1+1j, -1-1j])
    u = rng.choice(const, size=Nr)             # Random user symbols
    tau = 4.0                                  # Arbitrary spacing parameter
    data = qubo_from_vpp(H, u, tau, t=1)
    return H, u, tau, data
```

vpp_qubo_example.py

Function sample_and_build sets $N_r = N_t = 2$. It draws a random complex channel H with independent standard normal real and imaginary parts then divides by $\sqrt{2}$ so each complex coefficient has unit variance. It draws user symbols u from QPSK. It sets $\tau = 4.0$. It calls qubo_from_vpp to build Q offset C and decode. In the main script we call sample_and_build and then call brute_force_check. The script prints H, u, τ, Q, offset, both best energies, both perturbation vectors, and a boolean that shows whether the energies match.

vpp_qubo_example.py explained

```
import numpy as np
from vpp_qubo.vpp import sample_and_build, brute_force_check

# -------------------------
# Example 1
# -------------------------

# Generate random VPP instance with fixed seed for reproducibility
H, u, tau, data = sample_and_build(seed=3)

# sample_and_build returns:
#   H    : random 2x2 channel matrix (complex Gaussian entries)
#   u    : random user symbols (QPSK constellation)
#   tau  : spacing parameter (set = 4.0 inside sample_and_build)
#   data : dictionary containing QUBO info:
#          Q (matrix), offset (scalar), C (encoding matrix), decode (function)

# Check correctness using brute force:
#   1. Direct method: search all possible integer perturbations l
#   2. QUBO method  : search all possible binary vectors q
#   3. Compare resulting minimum energies and decoded perturbations
res = brute_force_check(H, u, tau, data["Q"], data["offset"],
                        data["C"], data["decode"])
```

vpp_qubo_example.py explained

```
# Print results for Example 1
print("\n")
print("Example 1: ")
print("Channel matrix H:\n", H)
print("User symbols u:", u)
print("Tau:", tau)
print("QUBO matrix Q:\n", data["Q"])
print("Offset:", data["offset"])
print("Direct best energy:", res["E_direct"])
print("QUBO best energy:", res["E_qubo"])
print("Decoded perturbation vector l* from QUBO:", res["l_from_qubo"])
print("Decoded perturbation vector l* direct:", res["l_direct"])

# Numerical tolerance: check if results match
# Here, 1e-8 was arbitrarily chosen as tolerance threshold
print("Match:", abs(res["E_direct"] - res["E_qubo"]) < 1e-8)
```

vpp_qubo_example.py explained

```
# -------------------------
# Example 2
# -------------------------

# Another random instance with different seed
H2, u2, tau2, data2 = sample_and_build(seed=4)

# Perform brute force check again
res2 = brute_force_check(H2, u2, tau2,
                         data2["Q"], data2["offset"],
                         data2["C"], data2["decode"])

# Print results for Example 2
print("\n")
print("Example 2: ")
print("Channel matrix H:\n", H2)
print("User symbols u:", u2)
print("Tau:", tau2)
print("QUBO matrix Q:\n", data2["Q"])
print("Offset:", data2["offset"])
print("Direct best energy:", res2["E_direct"])
print("QUBO best energy:", res2["E_qubo"])
print("Decoded perturbation vector l* from QUBO:", res2["l_from_qubo"])
print("Decoded perturbation vector l* direct:", res2["l_direct"])
print("Match:", abs(res2["E_direct"] - res2["E_qubo"]) < 1e-8)
```

How to read output

The printed channel H is the complex channel of size two by two. The vector u has two QPSK symbols. The scalar τ is the spacing. The QUBO matrix Q is real symmetric with size equal to the number of binary variables M. The offset is $y^T G y$. The direct best energy is the minimum of the original energy over integer v. The QUBO best energy is the minimum of $q^T Q q +$ offset over binary q. The decoded perturbation from QUBO and the best perturbation from the direct search should match on the toy examples which confirms the construction.

raw program output)

```
Example 1:
Channel matrix H:
 [[ 1.44314775-0.32007138j -1.80712807-0.15245022j]
 [ 0.29564053-1.42834589j -0.40147374-0.16400096j]]
User symbols u: [-1.+1.j -1.+1.j]
Tau: 4.0
QUBO matrix Q:
 [[ 1.87529499e+01 -1.05755497e+01 ... -7.90783412e+00]
  ...
 [-7.90783412e+00  3.95391706e+00 ...  1.49070452e+01]]
Offset: 1.2904706720279664
Direct best energy: 1.2904706720279664
QUBO best energy: 1.2904706720279664
Decoded perturbation vector l* from QUBO: [0 0 0 0]
Decoded perturbation vector l* direct: [0 0 0 0]
Match: True

Example 2:
Channel matrix H:
 [[-0.46088594-1.16064316j -0.12354378-0.00367926j]
 [ 1.17643052-0.44085544j  0.46608784+0.10509836j]]
User symbols u: [-1.+1.j -1.-1.j]
Tau: 4.0
QUBO matrix Q:
 [[ 2.93803688e+02 -1.40341844e+02 ...  1.20976166e+02]
```

Output Meaning

Example 1 and 2 results:

- **Channel matrix H:** A 2×2 complex matrix where each entry represents the gain of the channel between one transmit antenna and one user. Generated randomly using i.i.d. complex Gaussian samples (Rayleigh fading model).
- **User symbols u:** A length-2 vector, one QPSK symbol per user. For Example 1, $u = [-1 + j, -1 + j]$.
- **Tau (τ):** Spacing parameter used for perturbations. Here fixed to $\tau = 4.0$.
- **QUBO matrix Q:** The quadratic binary optimization matrix. It encodes the cost function in binary form. Large numbers come from inner products of basis vectors; near-zero values are numerical rounding errors.
- **Offset:** Constant term in the cost function. Both Q and offset are needed to evaluate the QUBO energy.
- **Direct best energy:** Minimum energy found by brute-force search over integer perturbations l.

Output meaning

Example 1 and 2 results:

- **QUBO best energy:** Minimum energy found by brute-force search over binary vectors q.
- **Decoded perturbation vector l^*:** The integer perturbation that minimizes the energy. Both methods produced $[0, 0, 0, 0]$.
- **Match = True:** Confirms that the QUBO formulation exactly matches the original mathematical formulation for these examples.

Assumptions in toy implementation

- Restricted examples to 2×2 channels to keep exhaustive checks feasible
- Generated H with independent standard normal real and imaginary parts, scaled by $1/\sqrt{2}$ so each complex coefficient has unit variance
- Used QPSK for u
- Set $\tau = 4.0$ as a simple spacing for the example
- Used two's complement with $t + 1$ bits per integer
- Limited brute force QUBO search to at most 26 binary variables to avoid explosive runtime
- Validated derivation by comparing direct and QUBO searches on small cases

Why the QUBO is quadratic

The norm squared of a linear function is a quadratic function. We have $\|F(y+\tau v)\|_2^2 = (y+\tau v)^T G(y+\tau v)$ where $G = F^T F$. This is quadratic in v. After encoding $v = Cq$ it becomes quadratic in q. Therefore a QUBO form is natural for this problem.

Why fold the linear term into the diagonal

For binary variables we have $q_i^2 = q_i$. A linear term $b_i q_i$ can be written as $b_i q_i^2$ and added to the diagonal entry Q_{ii}. This keeps the objective in pure quadratic form $q^T Q q$. The constant offset remains unchanged.

Receivers and the perturbation

Users do not need to know the explicit vector v if the receiver applies a modulo mapping that wraps symbols back into the original constellation region. The transmitter chooses v to reduce power. The receivers then map the received symbol into the basic region. This is the standard idea in vector perturbation schemes.

Why QPSK and why tau equals four

QPSK is the simplest complex constellation with nonzero real and imaginary parts and gives a clean toy example. The spacing parameter τ must be positive and of suitable scale relative to the constellation. I chose $\tau = 4.0$ for clarity in the example. Other values are possible and would change numeric entries in Q but not the derivation.

Channel sampling in the toy example

In rich scattering environments a complex baseband channel coefficient is often modeled as complex Gaussian with zero mean and independent real and imaginary parts. I draw H with independent standard normal real and imaginary entries and divide by $\sqrt{2}$ so that each complex coefficient has unit variance. This produces a Rayleigh fading model at the magnitude level. This choice is only for the toy script. The derivation does not depend on it.

Effect of $t = 1$ in Binary Encoding

The parameter t controls the number of magnitude bits for each integer component of the perturbation vector. When $t = 0$, each real component uses one bit. When $t = 1$, each component uses one sign bit and one magnitude bit, doubling the number of binary variables:

$$M = (t + 1) \times 2N_r.$$

For a 2×2 complex channel, $2N_r = 4$ and $M = 8$. Thus the QUBO matrix becomes 8×8. This is expected and indicates an expanded search range of integer values.

Reading the Output in Plain English

Channel and Symbols:

$$H \in \mathbb{C}^{2\times 2}, \quad u = [-1+j,\ -1+j], \quad \tau = 4.$$

H represents the downlink channel, u are QPSK symbols, and τ defines the lattice spacing.
QUBO matrix Q: An 8×8 matrix defining the quadratic energy function

$$E(q) = q^T Q q + \text{offset}.$$

The entries encode binary interactions between all bits of the realified perturbation vector.

Direct and QUBO Verification Methods

Two independent checks confirm the QUBO formulation:

1. **Direct method:** Evaluate

$$E(v) = \|F(y + \tau v)\|_2^2$$

   for integer vectors v directly using the real-valued model.

2. **QUBO method:** Encode $v = Cq$, minimize

$$q^T Qq + \text{offset},$$

   then decode $q \mapsto v$.

Matching energies and decoded vectors verify that the QUBO correctly reproduces the classical problem.

Example Results for $t = 1$

For both random 2×2 channel examples:

$$\text{Direct best energy} = \text{QUBO best energy},$$

and

$$l^*_{\text{QUBO}} = l^*_{\text{direct}} = [0, 0, 0, 0].$$

The perturbation vector is zero, meaning no integer offset reduced power further. Decoded perturbation vector is typically this because in random small channels, zero perturbation already minimizes power. “Match: True” confirms perfect equivalence of both formulations.

Summary

- Setting $t = 1$ doubles the binary variable count, yielding an 8×8 QUBO matrix.
- The dimension increase is correct and expected due to realification and two-bit encoding.
- The two verification paths—direct computation and QUBO minimization—produce identical results.
- This confirms the consistency and correctness of the VPP-to-QUBO mapping for the realified system.

Overview: From Complex Model to QUBO

1. Complex domain formulation:

$$x = P(u + \tau v), \quad v \in \mathbb{Z}[j]^{N_r}.$$

Minimize transmit power $\|P(u + \tau v)\|_2^2$ by choosing integer perturbation v.

2. Realification: Convert complex quantities into real form:

$$\phi(u) = \begin{bmatrix} \Re(u) \\ \Im(u) \end{bmatrix}, \quad \Phi(P) = \begin{bmatrix} \Re(P) & -\Im(P) \\ \Im(P) & \Re(P) \end{bmatrix}.$$

After this step, $v \in \mathbb{Z}^{2N_r}$.

3. Binary encoding: Represent each integer as $v = Cq$ with two's complement encoding. Parameter t controls the number of bits per integer:

$$M = (t+1) \cdot 2N_r, \quad q \in \{0,1\}^M.$$

4. QUBO formulation: Substitute $v = Cq$ into the quadratic energy:

$$E(q) = q^T Q q + \text{offset}, \quad Q = \tau^2 C^T G C + 2\tau C^T G y.$$

Minimizing this QUBO yields the same optimal perturbation as direct minimization in v.

Understanding the Parameter t

The parameter t controls the **binary precision** used to represent each integer variable in the perturbation vector v.

1. Purpose: t determines how many bits we use to describe the possible integer values of v_i. Each v_i is encoded using **two's complement** representation.

2. Bits used: Each integer variable uses $(t+1)$ bits:

$$\text{Weights: } [-2^t, 2^0, 2^1, \ldots, 2^{t-1}].$$

3. Representable range:

$$v_i \in [-2^t,\ 2^t - 1].$$

Example:

- $t = 0$: weights $[-1] \to$ integers $\{0, -1\}$, 1 bit per variable.
- $t = 1$: weights $[-2, 1] \to$ integers $\{-2, -1, 0, 1\}$, 2 bits per variable.
- $t = 2$: weights $[-4, 1, 2] \to$ integers $\{-4, -3, -2, -1, 0, 1, 2, 3\}$, 3 bits per variable.

4. Impact: Increasing t increases the **search range and resolution** of the perturbation vector, and therefore increases the **QUBO size**:

$$M = (t+1) \times 2N_r.$$

Solving the QUBO with PySA

Once we have Q and the offset, we can run a quantum-inspired simulated annealer such as PySA. The goal is to find a binary vector q that minimizes

$$E(q) = q^T Qq + \text{offset}.$$

PySA imitates thermal cooling: it starts from random q, perturbs bits, and accepts changes according to a Boltzmann probability. This gradually lowers energy and tends to find near-optimal solutions.
The solver outputs many candidate states and their corresponding energies. Each state is a binary vector q, which can be decoded to an integer vector $v = Cq$.

Annealing Parameters Explained

- **n_sweeps** — number of passes through all bits during cooling. More sweeps give slower, deeper convergence.
- **n_reads** — number of independent annealing runs. Each run starts from a random state and yields one energy sample.
- **n_replicas** — number of parallel temperature chains. Replicas allow hot (exploratory) and cold (precise) searches simultaneously.
- **min_temp**, **max_temp** — define the temperature range. The system cools logarithmically between these extremes.

Lower minimum temperature and more sweeps produce higher accuracy but require longer runtime.

Interpreting the Annealing Output

After annealing, we obtain:

- **Best energy:** the smallest $E(q)$ observed.
- **Best state:** the binary vector achieving that energy.
- **Decoded vector:** $v = Cq$, the corresponding perturbation.
- **Success rate:** fraction of samples whose energy equals the known ground-state energy (from brute-force check on small problems).

A high success rate means the annealer reliably reaches the global optimum. A low rate indicates the temperature schedule or number of sweeps should be adjusted.

Brute-Force Baseline for This Instance

Before running any annealing, we compute the exact ground state since the QUBO is small enough (24 variables). This gives a reference to evaluate annealing quality.

- **QUBO size:** 24×24
- **Ground-state energy (raw):**

$$E_{\mathrm{gs}} = -47.3846818$$

- **Ground-state physical energy:**

$$E_{\mathrm{phys}} = 9.32183926$$

- **Correct perturbation vector:**

$$\ell^{\star} = [\,0, 0, 0, 0, -1, -1, 0, 1, 0, 0, 0, 0\,]$$

This exact value allows direct computation of the success probability:

$$P_{\mathrm{hit}} = \frac{\#\{\text{samples with } E = E_{\mathrm{gs}}\}}{\text{total samples}}.$$

SA Sweep: Probability of Hitting Ground State

We performed a fixed-temperature SA sweep over six orders of magnitude. Your results:

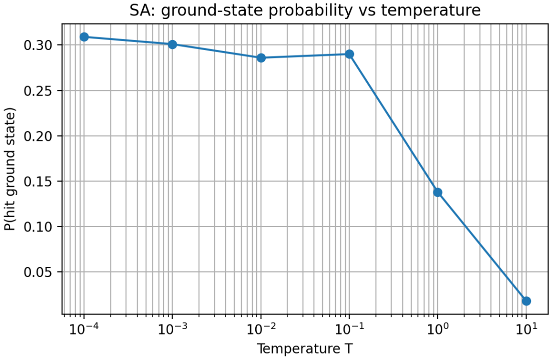
T	P_{hit}
10^{-4}	0.309
10^{-3}	0.301
10^{-2}	0.286
10^{-1}	0.290
1	0.138
10	0.018

- Maximum success at low temperature:

$$T_{\text{best}} = 10^{-4}, \qquad P_{\text{hit}} \approx 0.31.$$

- High temperatures destroy structure and give poor results.
- Very low temperatures behave like greedy descent, explaining why the best performance occurs there.

SA Sweep: Probability of Hitting Ground State



Parallel Tempering Grid Search

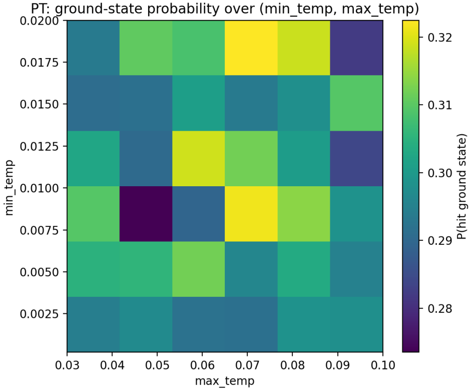
We sweep $(T_{\text{min}}, T_{\text{max}})$ over a 6×6 grid. For each pair, we run 1000 PT samples and compute the hit probability.

$$(T_{\text{min}}, T_{\text{max}}) \in [0.0002, 0.02] \times [0.03, 0.1].$$

Your results show:

- Best pair:
  $$T^{\star}_{\text{min}} = 0.02, \qquad T^{\star}_{\text{max}} = 0.072,$$
  achieving
  $$P_{\text{hit}} = 0.3225.$$
- This improves only slightly over pure SA, indicating a narrow global minimum basin.

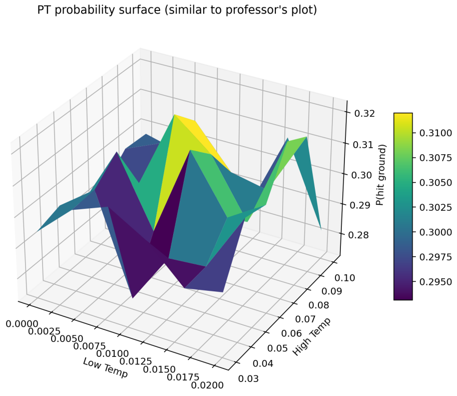
Parallel Tempering Grid Search



3D Probability Surface

We also generate a full 3D surface of the PT landscape. This allows visualization of the local ridge where PT performs best.

3D Probability Surface



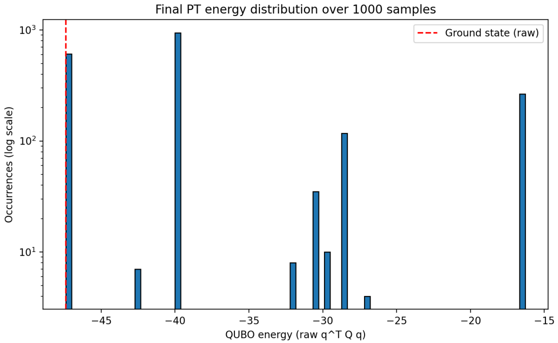
Key observation: the surface is relatively flat. No parameter choice exceeds $P_{\text{hit}} \approx 0.32$.

Final PT Run at Best Parameters

At $(T_{\min}, T_{\max}) = (0.02, 0.072)$:

- **Final PT hit rate:**
$$P_{\text{hit}} = 0.303.$$
- **Best observed energy:** -47.3846818 (matches ground truth).
- **Decoded vector:**
$$\ell_{\mathrm{PT}} = [\,0, 0, 0, 0, -1, -1, 0, 1, 0, 0, 0, 0\,]$$
- **Greedy polish:** no improvement beyond the raw samples, confirming PT already finds the global solution.

Final PT Run at Best Parameters



Problem Setup: QPSK VPP (4×4)

- Downlink VPP formulated as a QUBO
- Modulation: QPSK
- System size: $N_t = N_r = 4 \Rightarrow 16$ binary variables
- Channel matrix H drawn from complex Gaussian
- User symbols u fixed (downlink assumption)
- Perturbation spacing parameter τ varied

QUBO Construction Validation

- QUBO matrix size: 16×16
- QUBO is symmetric (as required)
- Energy consistency check:

$$v^T G v \; = \; q^T Q q + \text{offset}$$

- Brute-force physical energy and QUBO energy match numerically
- Decoded perturbation vector from QUBO equals brute-force optimum

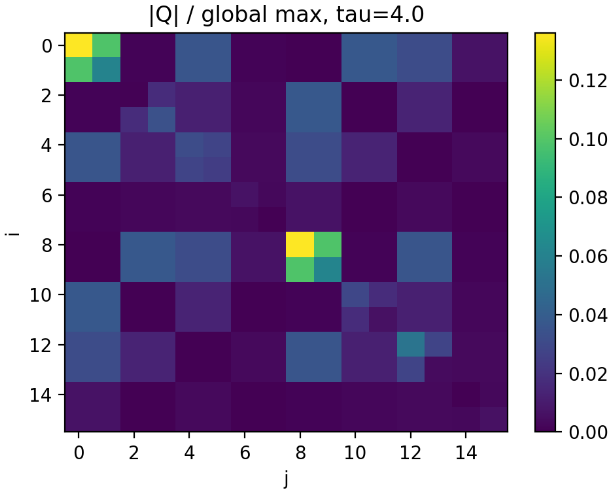
Annealing Difficulty at Small τ

- Tested $\tau = 4.0$ with:
  - $n_{\text{sweeps}} = 50$
  - $n_{\text{reads}} = 1000$
  - Parallel tempering with 2 replicas
- Single-temperature SA: 0% ground-state hit rate
- Parallel tempering grid: 0% ground-state hit rate
- Confirms VPP QUBO is significantly harder than MIMO detection

Motivation for QUBO Visualization

- Professor question:
  Is QPSK VPP structurally similar to QPSK MIMO detection or closer to 16QAM?
- In QPSK detection:
  - Strong diagonal (linear) terms
  - Weak off-diagonal (quadratic) terms
- In 16QAM detection:
  - Strong off-diagonal coupling
- Goal: quantify diagonal vs off-diagonal strength in VPP QUBO

QUBO Heatmap Visualization (QPSK VPP)



- Absolute value of QUBO matrix
- Normalized by global maximum
- Clear presence of strong off-diagonal terms

Diagonal vs Off-Diagonal Metrics

τ	max diag	max off-diag	ratio (off/diag)
1.0	30.9	12.3	0.40
2.0	86.3	49.0	0.57
4.0	270.7	196.2	0.72
8.0	933.8	784.7	0.84
12.0	1989.2	1765.5	0.89

Key Observation: QPSK VPP Is Not Diagonally Dominant

- Even for QPSK, off-diagonal terms are large
- Strongest off-diagonal term is:
  - 40% of diagonal at $\tau = 1$
  - Nearly 90% of diagonal at $\tau = 12$
- This is fundamentally different from QPSK MIMO detection
- QPSK VPP QUBO structurally resembles high-order modulation

Effect of τ on QUBO Structure

- Increasing τ scales both diagonal and off-diagonal terms
- Relative coupling strength increases with τ
- QUBO structure remains similar across τ
- Large τ improves solver hit rate but changes problem meaning

Interpretation of τ

- $\tau = 12$ gives 100% hit rate
- But produces unrealistically large spacing
- $\tau = 4$:
  - Conventional VPP spacing
  - 30% hit rate is reasonable given problem hardness
- $\tau = 1$:
  - Conceptually attractive
  - Poor numerical conditioning, solver failure

Why $\tau = 4$ Is the Conventional VPP Spacing (QPSK)

- In vector perturbation precoding, the transmitted symbols are

$$d = u + \tau\, l, \quad l \in \mathbb{Z}^{N_r}$$

- The receiver applies a modulo-τ operation to recover u
- To preserve correct decision regions, τ must match the constellation geometry
- Standard VPP literature chooses

$$\tau = 2\left(|c_{\max}| + \frac{\Delta}{2}\right)$$

  where $|c_{\max}|$ is the maximum real-axis symbol magnitude and Δ is the constellation spacing
- For QPSK: real symbols $\{\pm 1\}$, so $|c_{\max}| = 1$, $\Delta = 2$

$$\Rightarrow \tau = 2(1+1) = 4$$

- Therefore, $\tau = 4$ is the conventional and physically meaningful VPP spacing for QPSK

Why $\Delta = 2$ and $c_{\max} = 1$ for QPSK

- Vector perturbation spacing τ is defined per real dimension
- QPSK constellation:

$$\mathcal{S} = \{\pm 1 \pm j\}$$

- Real-axis projection:

$$\Re\{\mathcal{S}\} = \{-1, +1\}$$

- Maximum absolute real value:

$$c_{\max} = \max |\Re\{s\}| = 1$$

- Minimum spacing between adjacent real symbols:

$$\Delta = |(+1) - (-1)| = 2$$

- VPP spacing is chosen to match real-axis decision regions, not Euclidean distance

Summary of Answers to Professor's Questions

- QUBO is symmetric and correctly constructed
- QPSK VPP QUBO has strong off-diagonal terms
- Behavior is closer to 16QAM detection than QPSK detection
- τ controls energy scale, not coupling structure
- Large τ is numerically easy but not meaningful for VPP
- Small τ is meaningful but computationally challenging

QPSK MIMO Detection QUBO: Baseline

- Measured diagonal/off-diagonal coupling (QPSK detection):
- $\max(|Q_{\text{off}}|)/\max(|Q_{\text{diag}}|) = 0.27$
- $\text{mean}(|Q_{\text{off}}|)/\text{mean}(|Q_{\text{diag}}|) = 0.14$
- Diagonal (linear) terms dominate
- Confirms why QPSK MIMO detection is an easy optimization problem

16QAM VPP QUBO: Diagonal vs Off-Diagonal

- Measured coupling ratios for 16QAM VPP:
- $\tau = 4$: $\max(\text{off}/\text{diag}) = 0.38$
- $\tau = 8$: $\max(\text{off}/\text{diag}) = 0.55$
- $\tau = 12$: $\max(\text{off}/\text{diag}) = 0.65$
- Off-diagonal (quadratic) terms are no longer negligible
- Confirms known difficulty of higher-order modulation VPP

Why VPP Is Harder Than QPSK Detection

- QPSK MIMO detection:
  - $\max(\text{off}/\text{diag}) \approx 0.27$
- 16QAM VPP:
  - $\max(\text{off}/\text{diag}) = 0.38 \sim 0.65$
- VPP introduces significantly stronger quadratic coupling
- Optimization difficulty is driven by coupling structure, not modulation alone

Effect of τ on VPP QUBO Coupling

- Increasing τ increases absolute QUBO coefficients
- Increasing τ increases off-diagonal dominance:
  - $\tau = 4$: moderate coupling
  - $\tau = 8$: strong coupling
  - $\tau = 12$: very strong coupling
- τ sharpens the energy landscape but does not change QUBO structure
- Explains higher solver hit rate at large τ

Interim Conclusions (Structure Only)

- QPSK MIMO detection QUBO is diagonally dominant and easy
- VPP introduces strong quadratic coupling even for moderate τ
- 16QAM VPP confirms expected strong off-diagonal structure
- Large τ increases coupling and solver success, but is not physically motivated
- Final τ selection must be based on BER performance

BER Simulation Goal (Conventional VPP Baseline)

- Purpose: select τ using BER (not hit-rate) under a conventional VPP (vector perturbation precoding) chain.
- Downlink MIMO model:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \qquad \mathbf{n} \sim \mathcal{CN}(\mathbf{0}, \sigma^2 \mathbf{I})$$

- Data symbols (QPSK) per user:

$$\mathbf{u} \in \mathcal{S}^{N_r}, \quad \mathcal{S} = \{\pm 1 \pm j\}$$

- VPP idea: transmit a *perturbed* vector

$$\mathbf{d} = \mathbf{u} + \tau \mathbf{l}, \qquad \mathbf{l} \in (\mathbb{Z} + j\mathbb{Z})^{N_r}$$

  choosing $\mathbf{l}$ to reduce required transmit power.

Channel Generation: Rayleigh Fading (random_channel)

- Code:

$$\mathbf{H} = \frac{1}{\sqrt{2}}(\mathbf{H}_{\Re} + j\mathbf{H}_{\Im}), \quad (\mathbf{H}_{\Re})_{ij}, (\mathbf{H}_{\Im})_{ij} \sim \mathcal{N}(0,1)$$

- This yields i.i.d. entries $H_{ij} \sim \mathcal{CN}(0,1)$ (unit average power per complex coefficient).
- Justification: standard flat Rayleigh fading model; scaling by $1/\sqrt{2}$ ensures

$$\mathbb{E}[|H_{ij}|^2] = 1.$$

ZF Precoder Definition (zf_precoder)

- ZF precoding aims for $\mathbf{HP} \approx \mathbf{I}$.
- For square/full-rank $\mathbf{H} \in \mathbb{C}^{N_r \times N_t}$ with $N_r = N_t$:

$$\mathbf{P}_{\mathrm{ZF}} = \mathbf{H}^{-1}.$$

- More generally (if implemented via pseudo-inverse):

$$\mathbf{P}_{\mathrm{ZF}} = \mathbf{H}^{\mathrm{H}} \left(\mathbf{H}\mathbf{H}^{\mathrm{H}}\right)^{-1}.$$

- In this simulation, $N_r = N_t = 4$, so $\mathbf{P}$ behaves like an inverse/pseudo-inverse.

AWGN Model and SNR Convention (awgn_like)

- Code defines SNR relative to the average received (noiseless) energy:

$$E_s = \frac{1}{N_r}\sum_{k=1}^{N_r} |y_{k,\mathrm{noiseless}}|^2$$

- Converts $\mathrm{SNR}_{\mathrm{dB}}$ to linear:

$$\mathrm{SNR} = 10^{\mathrm{SNR}_{\mathrm{dB}}/10}.$$

- Sets noise spectral density:

$$N_0 = \frac{E_s}{\mathrm{SNR}}.$$

- For complex noise $\mathbf{n} \sim \mathcal{CN}(0, \sigma^2 \mathbf{I})$:

$$\sigma^2 = N_0, \quad n_k = \sigma \frac{(w_{\Re} + j w_{\Im})}{\sqrt{2}}, \quad w_{\Re}, w_{\Im} \sim \mathcal{N}(0,1)$$

  which matches the code:

$$\sigma = \sqrt{\frac{N_0}{2}} \text{ per real dimension.}$$

Bit Mapping for QPSK (bits_qpsk, bits_from_vec)

- QPSK symbol decision regions by quadrants.
- The code maps each symbol s to two bits:

$$b_0 = \mathbb{1}\{\Re(s) \geq 0\}, \qquad b_1 = \mathbb{1}\{\Im(s) \geq 0\}.$$

- Vector-to-bitstream:

$$\mathbf{b} = [b_{0,1}, b_{1,1}, b_{0,2}, b_{1,2}, \ldots, b_{0,N_r}, b_{1,N_r}]^{\mathsf{T}}.$$

- Justification: quadrant sign test is consistent with Gray-like labeling for $\{\pm 1 \pm j\}$ (up to bit ordering).

Conventional VPP Objective (brute_force_l_min_power)

- Perturbation search chooses $\mathbf{l}$ to minimize transmit power under ZF:

$$\mathbf{d}(\mathbf{l}) = \mathbf{u} + \tau\mathbf{l}, \qquad \mathbf{x}_{\text{unnorm}}(\mathbf{l}) = \mathbf{P}\mathbf{d}(\mathbf{l})$$

$$P_t(\mathbf{l}) = \|\mathbf{x}_{\text{unnorm}}(\mathbf{l})\|_2^2.$$

- Optimization solved by bounded search:

$$\mathbf{l}^\star = \arg\min_{\mathbf{l}\in\mathcal{L}} \|\mathbf{P}(\mathbf{u} + \tau\mathbf{l})\|_2^2$$

  with

$$\mathcal{L} = \{-1, 0, 1\}^{2N_r} \Rightarrow l_k = a_k + jb_k, \;\; a_k, b_k \in \{-1, 0, 1\}.$$

- Justification: provides a conventional reference baseline (exact over the bounded set) for 4×4.

Why the Search Space is 3^{2N_r}

- The code enumerates integer candidates for real and imaginary parts separately:

$$\mathbf{l} = \mathbf{a} + j\mathbf{b}, \quad \mathbf{a}, \mathbf{b} \in \{-1, 0, 1\}^{N_r}.$$

- Total combinations:

$$|\mathcal{L}| = 3^{N_r} \cdot 3^{N_r} = 3^{2N_r}.$$

- For $N_r = 4$:

$$|\mathcal{L}| = 3^8 = 6561,$$

  feasible for BER sweeps.
- Justification: this is a practical bounded approximation to the infinite lattice search used in theoretical VP formulations.

Transmit Normalization and Its Role (simulate_ber)

- After selecting $\mathbf{l}^\star$:

$$\mathbf{d} = \mathbf{u} + \tau \mathbf{l}^\star, \quad \mathbf{x}_{\text{unnorm}} = \mathbf{P}\mathbf{d}, \quad P_t = \|\mathbf{x}_{\text{unnorm}}\|_2^2.$$

- Code transmits normalized:

$$\mathbf{x} = \frac{\mathbf{x}_{\text{unnorm}}}{\sqrt{P_t}} = \frac{\mathbf{P}\mathbf{d}}{\sqrt{P_t}}.$$

- Then noiseless received vector:

$$\mathbf{y}_0 = \mathbf{H}\mathbf{x} = \frac{\mathbf{H}\mathbf{P}\mathbf{d}}{\sqrt{P_t}} \approx \frac{\mathbf{d}}{\sqrt{P_t}}.$$

- Justification: isolates BER vs SNR fairly by fixing average transmit power per symbol vector.

Receiver Processing: Undo Scale, Modulo, Slice

- After AWGN:

$$\mathbf{y} = \mathbf{y}_0 + \mathbf{n}.$$

- Code undoes the scalar normalization:

$$\mathbf{z} = \mathbf{y}\sqrt{P_t} \approx \mathbf{d} + \mathbf{n}', \quad \mathbf{n}' = \sqrt{P_t}\mathbf{n}.$$

- Apply modulo-τ (componentwise) to remove perturbation:

$$\mathbf{z}_{\text{mod}} = \gamma_\tau(\mathbf{z}).$$

- Ideal modulo property (noise-free):

$$\gamma_\tau(\mathbf{u} + \tau\mathbf{l}) = \mathbf{u}.$$

- Finally, symbol slicing:

$$\hat{\mathbf{u}} = Q_{\mathcal{S}}(\mathbf{z}_{\text{mod}})$$

where $Q_{\mathcal{S}}(\cdot)$ maps to nearest constellation point.

Modulo Operator (What gamma_tau Represents)

- $\gamma_\tau(\cdot)$ is a componentwise modulo mapping that wraps the real and imaginary parts into a fundamental region.
- Conceptually (per dimension), for a real scalar x:

$$\gamma_\tau(x) = x - \tau \cdot \mathrm{round}\left(\frac{x}{\tau}\right)$$

  (or an equivalent centered modulo mapping; exact convention depends on your implementation).
- Applied separately to $\Re(\cdot)$ and $\Im(\cdot)$:

$$\gamma_\tau(z) = \gamma_\tau(\Re z) + j\gamma_\tau(\Im z).$$

- Justification: VPP relies on modulo at the receiver to cancel $\tau\mathbf{l}$ without explicitly knowing $\mathbf{l}$.

BER Computation (simulate_ber)

- For each frame:

$$\mathbf{b} = \mathsf{bits}(\mathbf{u}), \quad \hat{\mathbf{b}} = \mathsf{bits}(\hat{\mathbf{u}}).$$

- Accumulate bit errors:

$$N_e \leftarrow N_e + \sum_i \mathbb{1}\{b_i \neq \hat{b}_i\}, \qquad N_b \leftarrow N_b + |\mathbf{b}|.$$

- BER estimate at each (τ, SNR):

$$\mathrm{BER}(\tau, \mathsf{SNR}) = \frac{N_e}{N_b}.$$

- Outer loops produce BER curves:

$$\{\mathrm{BER}(\tau, \mathsf{SNR}_k)\}_{k=1}^{K} \quad \text{for each } \tau.$$

Why This Answers the Professor's τ Question

- Professor: choose τ using BER under conventional VPP.
- This code measures exactly:

  $$\tau \mapsto \mathrm{BER}(\tau, \mathsf{SNR})$$

  using:
  - a conventional VPP perturbation selection (min P_t over a bounded integer set),
  - a correct downlink chain (ZF precoding, AWGN),
  - the standard receiver (scale undo, modulo-τ, slicing).
- Output: pick τ that minimizes BER over the SNR region of interest, rather than maximizing solver hit-rate.

Script Structure and Reproducibility

- Fixed RNG seed $\rightarrow$ reproducible results.
- Single channel realization in current code:

$$\mathbf{H} \text{ is fixed for all frames.}$$

- Justification: isolates τ effect on BER for a given channel.

BER vs. τ (4×4 QPSK, Conventional VPP Baseline)

τ	0 dB	5 dB	10 dB	15 dB	20 dB
1	0.502	0.492	0.502	0.498	0.497
2	0.503	0.510	0.499	0.495	0.494
4	0.451	0.261	0.063	0.0029	0
8	0.291	0.175	0.0696	0.0178	0.00163
12	0.184	0.0546	0.0115	0.0055	0.000375

BER-Based Conclusions for τ Selection

$\tau = 1$ and $\tau = 2$ give BER $\approx$ 0.5 at all SNR $\Rightarrow$ effectively random decisions after modulo/slicing. For moderate-to-high SNR (10–20 dB), $\tau = 4$ is best in this experiment:

$$\mathrm{BER}(10\ \text{dB}) = 6.3{\times}10^{-2},\ \mathrm{BER}(15\ \text{dB}) = 2.9{\times}10^{-3},\ \mathrm{BER}(20\ \text{dB}) = 0$$

For low SNR (0–5 dB), larger τ helps; $\tau = 12$ is best here:

$$\mathrm{BER}(5\ \text{dB}) = 5.46 \times 10^{-2}$$

Therefore, BER-optimal τ depends on the operating SNR; $\tau = 4$ is the best conventional choice for the high-SNR regime.

Practical τ Range from BER (QPSK, 4×4 VPP)

- BER results show a clear threshold behavior in τ
- For $\tau < 4$:
  - BER ≈ 0.5 for all SNR
  - Modulo operation causes severe ambiguity
  - $\tau = 1$ and $\tau = 2$ are not viable in this setup
- For $\tau \geq 4$:
  - BER decreases with SNR as expected
  - $\tau = 4$ gives best performance at moderate/high SNR
  - Larger τ improves low-SNR robustness but degrades high-SNR optimality
- Conclusion: for QPSK VPP, a physically meaningful and usable range is

$$\tau \in [4,\ 8] \quad \text{(depending on operating SNR)}$$

Question 1: Is $\tau = 12$ Meaningful?

- Conventional VPP theory (QPSK):

$$c_{\max} = 1, \quad \Delta = 2 \;\Rightarrow\; \tau = 4$$

- BER results (4×4 QPSK VPP):
  - Low SNR:
    - 5 dB: $\tau = 12$ BER $= 5.46 \times 10^{-2}$ vs $\tau = 4$ BER $= 2.61 \times 10^{-1}$
  - Moderate/High SNR:
    - 15 dB: $\tau = 4$ BER $= 2.88 \times 10^{-3}$ vs $\tau = 12$ BER $= 5.50 \times 10^{-3}$
    - 20 dB: $\tau = 4$ BER $= 0$ vs $\tau = 12$ BER $= 3.75 \times 10^{-4}$
- Conclusion:
  - $\tau = 12$ improves low-SNR BER
  - $\tau = 4$ is optimal at moderate/high SNR and is the conventional choice

Question 2: 30% Hit Rate at $\tau = 4$ vs 100% at $\tau = 12$

- Hit-rate results (PySA):
  - $\tau = 4$: $\sim$30%
  - $\tau = 8$: $\sim$99.95%
  - $\tau = 12$: 100%
- BER comparison:
  - $\tau = 4$ outperforms $\tau = 12$ at 15–20 dB
- Interpretation:
  - Large τ simplifies QUBO and improves solver hit rate
  - Smaller τ preserves physical VPP spacing and improves BER
- Conclusion:
  - 30% hit rate at $\tau = 4$ is acceptable
  - Solver hit rate $\neq$ communication performance

Question 3: What Happens at $\tau = 1$?

- Hit rate:
  - $\tau = 1$: 0% (PySA)
- BER:
  - BER ≈ 0.5 for all SNR (0–20 dB)
- Interpretation:
  - Modulo-τ causes severe ambiguity
  - Slicing becomes random
- Conclusion:
  - $\tau = 1$ is not viable for QPSK VPP

Question 4: QUBO Structure (Symmetry)

- QUBO matrices are symmetric:

$$Q = Q^{\mathsf{T}}$$

- Verified by construction and visualization
- Not upper triangular

Question 5: Diagonal vs Off-Diagonal Strength (QPSK Detection)

- QPSK MIMO detection QUBO:
  - max off/diag $= 0.27$
  - mean off/diag $= 0.14$
- Strong diagonal dominance
- Explains why MIMO detection is easy and yields 100% success

Question 6: QPSK VPP QUBO Coupling Strength

- QPSK VPP diagonal/off-diagonal ratios:
  - $\tau = 1$: max off/diag ≈ 0.40
  - $\tau = 4$: max off/diag ≈ 0.72
  - $\tau = 8$: max off/diag ≈ 0.84
  - $\tau = 12$: max off/diag ≈ 0.89
- Interpretation:
  - Quadratic terms are strong even for QPSK
  - VPP QPSK is not diagonally dominant

Question 7: Is QPSK VPP More Like QPSK Detection or 16QAM?

- 16QAM VPP off/diag ratios:
  - $\tau = 4$: 0.38
  - $\tau = 8$: 0.55
  - $\tau = 12$: 0.65
- Comparison:
  - QPSK detection: 0.27
  - QPSK VPP: 0.72–0.89
  - 16QAM VPP: 0.38–0.65
- Conclusion:
  - QPSK VPP behaves more like 16QAM than QPSK detection

Question 8: Show Actual QUBO Values

- Provided explicitly:
  - Top-left sub-block of QUBO matrix
  - Largest off-diagonal entries
- Observation:
  - Off-diagonal magnitudes comparable to diagonal terms

Question 9: Should τ Be Small Since $\mathbf{u}$ Is Fixed?

- Intuition:
  - Smaller τ explores nearby perturbations
- Empirical result:
  - $\tau = 1, 2$: BER ≈ 0.5
- Conclusion:
  - τ must be large enough to cover decision regions
  - Too small τ breaks modulo recovery

Question 10: Required τ Range for VPP

- Theory (QPSK):

$$\tau = 2(c_{\text{max}} + \Delta/2) = 4$$

- BER results:
  - $\tau < 4$: unusable
  - $\tau = 4$: best at moderate/high SNR
  - $\tau = 8$: trade-off
  - $\tau = 12$: low-SNR improvement only
- Practical range:

$$\tau \in [4, 8]$$

Final Summary

- QPSK VPP produces strongly coupled QUBOs
- More similar to 16QAM than QPSK detection
- $\tau = 4$ is:
  - Theoretical (conventional)
  - BER-optimal at moderate/high SNR
- Large τ improves solver success but reduces physical meaning
- $\tau < 4$ fails completely