paper

October 7, 2025

[8]:
```
"""
Unified UC-ADMM runner (5 units × 6 hours, deterministic)
Compares six Block-2 strategies:
  • 3-QUBO + Brute Force
  • 3-QUBO + DVQE
  • Micro-QUBO + Brute Force
  • Micro-QUBO + DVQE
  • Batched-QUBO + Brute Force
  • Batched-QUBO + DVQE

Outputs ONE semilog residual plot with publication-friendly styling.
"""

import numpy as np
import cvxpy as cp
import math
import matplotlib.pyplot as plt
from itertools import product

# =============== Matplotlib: paper-friendly defaults ===============
plt.rcParams.update({
    "figure.figsize": (6.6, 3.6),  # two-column friendly
    "font.size": 9,
    "axes.labelsize": 10,
    "axes.titlesize": 10,
    "legend.fontsize": 8,
    "xtick.labelsize": 8,
    "ytick.labelsize": 8,
    "lines.linewidth": 2.0,
    "savefig.dpi": 300,
})

# =============== DVQE import (optional) ===============
try:
    from raiselab import DVQE
except Exception:
    DVQE = None
```

```
    print("Warning: raiselab.DVQE not found. DVQE variants will error if run.")

# =============== Shared problem (5×6) ===============
N, T = 5, 6
A = np.array([500, 600, 450, 400, 550], float)
B = np.array([20.0, 25.0, 18.0, 22.0, 21.0], float)
C = np.array([0.002, 0.0015, 0.0025, 0.002, 0.0018], float)
S = np.array([200, 200, 250, 150, 180], float)
H = np.array([0, 0, 0, 0, 0], float)

Pmin = np.array([10, 20, 15, 10, 20], float)
Pmax = np.array([60, 80, 70, 50, 85], float)

RU = np.array([30, 30, 25, 25, 30], float)
RD = np.array([30, 30, 25, 25, 30], float)
SU = np.array([40, 40, 35, 35, 40], float)
SD = np.array([40, 40, 35, 35, 40], float)

Umin = np.array([2, 2, 3, 2, 2], int)
Dmin = np.array([2, 2, 2, 2, 2], int)

L = np.array([150, 170, 180, 160, 140, 130], float)
R_up_req = np.array([10, 12, 12, 10, 8, 8], float)
R_dn_req = np.array([8, 8, 10, 10, 8, 6], float)
delta_tau = 1.0

# =============== ADMM hyperparameters (kept identical across runs)␣
  ↪===============
rho_y = rho_u = rho_v = 9.0e5
beta_y = beta_u = beta_v = 2.0e6
epsilon = 1e-3
max_iter = 4000

ACCEPT_TOL = 1e-12
Y_THRESHOLD = 0.5
RAMP_TOL = 1e-7

# Micro-QUBO penalties
gamma_c    = 0.20 * rho_y
gamma_ss   = 0.10 * rho_y
gamma_u2y  = 0.05 * rho_y
gamma_v2ny = 0.05 * rho_y
gamma_y = gamma_u = gamma_v = 0.10 * rho_y

# Batched-QUBO settings
K_BATCHES = 3
```

```
# DVQE settings (fixed across all DVQE runs)
dvqe_mode = "distributed"
dvqe_depth = 2
dvqe_lr = 0.1
dvqe_max_iters = 100
qpu_qubit_config_3qubo = [3, 3, 3, 3, 3]     # N bits per per-time QUBO
qpu_qubit_config_micro = [2, 2, 2, 2, 2, 2] # placeholder for tiny 3-bit micros␣
 ↪(monolithic)
qpu_qubit_config_batch = [3, 3, 3, 3, 3]     # small batches (<=5 units → <=15␣
 ↪bits)

# =============== Utilities ===============
def reset_state(seed=7):
    rng = np.random.default_rng(seed)
    p0 = np.ones(N)* 20
    y0 = (p0 > 0.0).astype(int)
    y = np.ones((N, T)) * 0.5
    u = np.zeros((N, T))
    v = np.zeros((N, T))
    p = np.tile(L / max(N, 1), (N, 1))
    r_up = np.zeros((N, T)); r_dn = np.zeros((N, T))
    z_y = np.zeros((N, T), int); z_u = np.zeros((N, T), int); z_v = np.
 ↪zeros((N, T), int)
    s_y = np.zeros((N, T)); s_u = np.zeros((N, T)); s_v = np.zeros((N, T))
    lam_y = np.zeros((N, T)); lam_u = np.zeros((N, T)); lam_v = np.zeros((N, T))
    return dict(p0=p0, y0=y0, y=y, u=u, v=v, p=p, r_up=r_up, r_dn=r_dn,
                z_y=z_y, z_u=z_u, z_v=z_v, s_y=s_y, s_u=s_u, s_v=s_v,
                lam_y=lam_y, lam_u=lam_u, lam_v=lam_v)

def solve_qubo_bruteforce(Q, q_linear):
    Q = np.asarray(Q, float); q = np.asarray(q_linear, float).ravel()
    n = q.size
    if Q.shape != (n, n): raise ValueError(f"Q shape {Q.shape} != ({n},{n})")
    Q = 0.5*(Q+Q.T)
    best_z, best_E = None, float("inf")
    for cand in product([0,1], repeat=n):
        z = np.fromiter(cand, float, n)
        E = z@Q@z + q@z
        if E < best_E:
            best_E = E; best_z = z.astype(int)
    return best_z, best_E

def micro_qubo_coeffs(qy, qu, qv, y_hat, u_hat, v_hat, y_ref):
    Q = np.zeros((3,3), float)  # [y,u,v]
    c = np.array([qy, qu, qv], float)
    # anchors
    c[0] += gamma_y*(1 - 2*y_hat)
```

```
    c[1] += gamma_u*(1 - 2*u_hat) + gamma_u2y
    c[2] += gamma_v*(1 - 2*v_hat)
    # soft local transition (c)
    c[0] += gamma_c*(1 - 2*y_ref)
    c[1] += gamma_c*(1 + 2*y_ref)
    c[2] += gamma_c*(1 - 2*y_ref)
    Q[0,1] += -2*gamma_c; Q[1,0]=Q[0,1]
    Q[0,2] += +2*gamma_c; Q[2,0]=Q[0,2]
    Q[1,2] += -2*gamma_c; Q[2,1]=Q[1,2]
    # u+v <=1
    Q[1,2] += gamma_ss; Q[2,1]=Q[1,2]
    # start -> on
    Q[0,1] += -gamma_u2y; Q[1,0]=Q[0,1]
    # shutdown -> off
    Q[0,2] += gamma_v2ny; Q[2,0]=Q[0,2]
    return Q, c

def local_augL(y_it,u_it,v_it, sy_it,su_it,sv_it, lamy,lamu,lamv, zy,zu,zv):
    Ly = lamy*(y_it - zy + sy_it) + 0.5*rho_y*(y_it - zy + sy_it)**2
    Lu = lamu*(u_it - zu + su_it) + 0.5*rho_u*(u_it - zu + su_it)**2
    Lv = lamv*(v_it - zv + sv_it) + 0.5*rho_v*(v_it - zv + sv_it)**2
    return float(Ly + Lu + Lv)

def block1_qp(state):
    y,u,v,p,r_up,r_dn = [cp.Variable((N,T)) for _ in range(6)]
    r_up.nonneg=True; r_dn.nonneg=True
    econ = (cp.sum(cp.multiply(A[:,None],y)) +
            cp.sum(cp.multiply(B[:,None],p)) +
            cp.sum(cp.multiply(C[:,None],cp.square(p))) +
            cp.sum(cp.multiply(S[:,None],u)) +
            cp.sum(cp.multiply(H[:,None],v)))
    pen_y = cp.sum(cp.multiply(state["lam_y"], y - state["z_y"] +␣
 ↪state["s_y"])) + (rho_y/2)*cp.sum_squares(y - state["z_y"] + state["s_y"])
    pen_u = cp.sum(cp.multiply(state["lam_u"], u - state["z_u"] +␣
 ↪state["s_u"])) + (rho_u/2)*cp.sum_squares(u - state["z_u"] + state["s_u"])
    pen_v = cp.sum(cp.multiply(state["lam_v"], v - state["z_v"] +␣
 ↪state["s_v"])) + (rho_v/2)*cp.sum_squares(v - state["z_v"] + state["s_v"])
    obj = cp.Minimize(econ + pen_y + pen_u + pen_v)
    cons = [y>=0, y<=1, u>=0, u<=1, v>=0, v<=1, p>=0]
    for t in range(T):
        cons += [cp.sum(p[:,t]) == L[t]]
    cons += [p >= cp.multiply(Pmin[:,None], y),
             p <= cp.multiply(Pmax[:,None], y)]
    cons += [y[:,0] - state["y0"] == u[:,0] - v[:,0],
             u[:,0] + v[:,0] <= 1]
    for t in range(1,T):
        cons += [y[:,t] - y[:,t-1] == u[:,t] - v[:,t],
```

```
                u[:,t] + v[:,t] <= 1]
    for i in range(N):
        Ui,Di = int(Umin[i]), int(Dmin[i])
        for t in range(T):
            k_up=max(0,t-Ui+1); k_dn=max(0,t-Di+1)
            cons += [cp.sum(u[i,k_up:t+1]) <= y[i,t]]
            cons += [cp.sum(v[i,k_dn:t+1]) <= 1 - y[i,t]]
    for i in range(N):
        cons += [p[i,0] - state["p0"][i] <= RU[i]*state["y0"][i] + SU[i]*u[i,0]]
        cons += [state["p0"][i] - p[i,0] <= RD[i]*y[i,0] + SD[i]*v[i,0]]
        for t in range(1,T):
            cons += [p[i,t] - p[i,t-1] <= RU[i]*y[i,t-1] + SU[i]*u[i,t]]
            cons += [p[i,t-1] - p[i,t] <= RD[i]*y[i,t] + SD[i]*v[i,t]]
    cons += [r_up <= cp.multiply(Pmax[:,None], y) - p,
             r_dn <= p - cp.multiply(Pmin[:,None], y),
             r_up <= RU[:,None]*delta_tau,
             r_dn <= RD[:,None]*delta_tau]
    for t in range(T):
        cons += [cp.sum(r_up[:,t]) >= R_up_req[t],
                 cp.sum(r_dn[:,t]) >= R_dn_req[t]]

    # warm-start
    y.value=state["y"]; u.value=state["u"]; v.value=state["v"]
    p.value=state["p"]; r_up.value=state["r_up"]; r_dn.value=state["r_dn"]

    prob = cp.Problem(obj, cons)
    installed = set(cp.installed_solvers())
    if "OSQP" in installed:
        prob.solve(solver=cp.OSQP, eps_abs=1e-8, eps_rel=1e-8, max_iter=800000,␣
↪polish=True, warm_start=True, verbose=False)
    if prob.status not in ("optimal","optimal_inaccurate") and "SCS" in␣
↪installed:
        prob.solve(solver=cp.SCS, eps=5e-8, max_iters=1_200_000,␣
↪warm_start=True, verbose=False)

    state["y"]=y.value; state["u"]=u.value; state["v"]=v.value
    state["p"]=p.value; state["r_up"]=r_up.value; state["r_dn"]=r_dn.value

def block3_and_duals(state):
    y,u,v = state["y"], state["u"], state["v"]
    z_y,z_u,z_v = state["z_y"], state["z_u"], state["z_v"]
    lam_y,lam_u,lam_v = state["lam_y"], state["lam_u"], state["lam_v"]
    s_y = -(lam_y + rho_y*(y - z_y)) / (beta_y + rho_y)
    s_u = -(lam_u + rho_u*(u - z_u)) / (beta_u + rho_u)
    s_v = -(lam_v + rho_v*(v - z_v)) / (beta_v + rho_v)
    s_y = np.maximum(0.0, s_y); s_u=np.maximum(0.0, s_u); s_v=np.maximum(0.0,␣
↪s_v)
```

```
    state["s_y"],state["s_u"],state["s_v"]=s_y,s_u,s_v
    state["lam_y"] = lam_y + 0.5*rho_y*(y - z_y + s_y)
    state["lam_u"] = lam_u + 0.5*rho_u*(u - z_u + s_u)
    state["lam_v"] = lam_v + 0.5*rho_v*(v - z_v + s_v)

def residual(state):
    y,u,v = state["y"], state["u"], state["v"]
    z_y,z_u,z_v = state["z_y"], state["z_u"], state["z_v"]
    s_y,s_u,s_v = state["s_y"], state["s_u"], state["s_v"]
    return math.sqrt(
        np.linalg.norm((y - z_y + s_y).ravel())**2 +
        np.linalg.norm((u - z_u + s_u).ravel())**2 +
        np.linalg.norm((v - z_v + s_v).ravel())**2
    )

# =============== Block-2 implementations ===============
def block2_3qubo_bruteforce(state):
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
 ↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
 ↪state["z_v"].copy()
    Q0 = np.zeros((N,N), float)
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        zy,_ = solve_qubo_bruteforce(Q0, qy)
        zu,_ = solve_qubo_bruteforce(Q0, qu)
        zv,_ = solve_qubo_bruteforce(Q0, qv)
        zy,zu,zv = map(lambda z:(np.asarray(z)>0.5).astype(int), (zy,zu,zv))
        state["z_y"][:,t] = zy if float(qy @ zy)+ACCEPT_TOL < float(qy @␣
 ↪z_y_prev[:,t]) else z_y_prev[:,t]
        state["z_u"][:,t] = zu if float(qu @ zu)+ACCEPT_TOL < float(qu @␣
 ↪z_u_prev[:,t]) else z_u_prev[:,t]
        state["z_v"][:,t] = zv if float(qv @ zv)+ACCEPT_TOL < float(qv @␣
 ↪z_v_prev[:,t]) else z_v_prev[:,t]

def block2_3qubo_dvqe(state):
    if DVQE is None: raise RuntimeError("DVQE not available")
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
 ↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
 ↪state["z_v"].copy()
    Q0 = np.zeros((N,N), float)
```

```
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        zy,_,_ = DVQE(mode=dvqe_mode, Q=Q0, q_linear=qy, init_type=2,
                      depth=dvqe_depth, lr=dvqe_lr, max_iters=dvqe_max_iters,
                      qpu_qubit_config=qpu_qubit_config_3qubo, rel_tol=1e-6)
        zu,_,_ = DVQE(mode=dvqe_mode, Q=Q0, q_linear=qu, init_type=2,
                      depth=dvqe_depth, lr=dvqe_lr, max_iters=dvqe_max_iters,
                      qpu_qubit_config=qpu_qubit_config_3qubo, rel_tol=1e-6)
        zv,_,_ = DVQE(mode=dvqe_mode, Q=Q0, q_linear=qv, init_type=2,
                      depth=dvqe_depth, lr=dvqe_lr, max_iters=dvqe_max_iters,
                      qpu_qubit_config=qpu_qubit_config_3qubo, rel_tol=1e-6)
        zy,zu,zv = map(lambda z:(np.asarray(z)>0.5).astype(int), (zy,zu,zv))
        state["z_y"][:,t] = zy if float(qy @ zy)+ACCEPT_TOL < float(qy @␣
↪z_y_prev[:,t]) else z_y_prev[:,t]
        state["z_u"][:,t] = zu if float(qu @ zu)+ACCEPT_TOL < float(qu @␣
↪z_u_prev[:,t]) else z_u_prev[:,t]
        state["z_v"][:,t] = zv if float(qv @ zv)+ACCEPT_TOL < float(qv @␣
↪z_v_prev[:,t]) else z_v_prev[:,t]

def block2_micro_bruteforce(state):
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
↪state["z_v"].copy()
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        for i in range(N):
            Q3,c3 = micro_qubo_coeffs(qy[i], qu[i], qv[i], y[i,t], u[i,t],␣
↪v[i,t],
                                      (z_y_prev[i,t-1] if t>0 else␣
↪state["y0"][i]))
            z3,_ = solve_qubo_bruteforce(Q3, c3)
            zy,zu,zv = map(int, np.asarray(z3).ravel())
            old = local_augL(y[i,t],u[i,t],v[i,t], s_y[i,t],s_u[i,t],s_v[i,t],
                             lam_y[i,t],lam_u[i,t],lam_v[i,t],␣
↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t])
            new = local_augL(y[i,t],u[i,t],v[i,t], s_y[i,t],s_u[i,t],s_v[i,t],
                             lam_y[i,t],lam_u[i,t],lam_v[i,t], zy,zu,zv)
            if new + ACCEPT_TOL <= old:
                state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
↪zy,zu,zv
```

```
            else:
                state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t]

def block2_micro_dvqe(state):
    if DVQE is None: raise RuntimeError("DVQE not available")
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
 ↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
 ↪state["z_v"].copy()
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        for i in range(N):
            Q3,c3 = micro_qubo_coeffs(qy[i], qu[i], qv[i], y[i,t], u[i,t],␣
 ↪v[i,t],
                                      (z_y_prev[i,t-1] if t>0 else␣
 ↪state["y0"][i]))
            z3,_,_ = DVQE(mode="monolithic", Q=Q3, q_linear=c3, init_type=2,
                          depth=2, lr=0.1, max_iters=100,␣
 ↪qpu_qubit_config=[2,2,2,2,2,2], rel_tol=1e-6)
            zy,zu,zv = map(int, np.asarray(z3).ravel())
            old = local_augL(y[i,t],u[i,t],v[i,t], s_y[i,t],s_u[i,t],s_v[i,t],
                             lam_y[i,t],lam_u[i,t],lam_v[i,t],␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t])
            new = local_augL(y[i,t],u[i,t],v[i,t], s_y[i,t],s_u[i,t],s_v[i,t],
                             lam_y[i,t],lam_u[i,t],lam_v[i,t], zy,zu,zv)
            if new + ACCEPT_TOL <= old:
                state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪zy,zu,zv
            else:
                state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t]

def hardness_score(Q, c):
    # simple hardness proxy (gap + coupling/field ratio)
    Q = np.asarray(Q,float); c = np.asarray(c,float)
    E = []
    for cand in product([0,1], repeat=3):
        z = np.array(cand,float)
        E.append(float(z@Q@z + c@z))
    E = sorted(E); delta = E[1]-E[0] if len(E)>1 else 1.0
    coup = abs(Q[0,1])+abs(Q[1,2])+abs(Q[0,2])
    field = abs(c[0])+abs(c[1])+abs(c[2])+1e-12
```

```
    return (1.0/(delta+1e-9)) + (coup/(field+1.0))

def batch_partition(Q_list, c_list, k=K_BATCHES):
    items = [(i, hardness_score(Q_list[i], c_list[i])) for i in␣
 ↪range(len(Q_list))]
    items.sort(key=lambda x: x[1], reverse=True)
    k = max(1, min(k, len(Q_list)))
    groups = [[] for _ in range(k)]; loads = [0.0]*k
    for i,score in items:
        b = int(np.argmin(loads))
        groups[b].append(i); loads[b]+=score
    return groups

def block2_batched_bruteforce(state):
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
 ↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
 ↪state["z_v"].copy()
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        Q_list=[]; c_list=[]
        for i in range(N):
            Q3,c3 = micro_qubo_coeffs(qy[i], qu[i], qv[i], y[i,t], u[i,t],␣
 ↪v[i,t],
                                      (z_y_prev[i,t-1] if t>0 else␣
 ↪state["y0"][i]))
            Q_list.append(Q3); c_list.append(c3)
        groups = batch_partition(Q_list, c_list, k=K_BATCHES)
        for units in groups:
            m=len(units); Qb=np.zeros((3*m,3*m)); cb=np.zeros(3*m)
            for k,i in enumerate(units):
                idx=slice(3*k,3*k+3); Qb[idx,idx]=Q_list[i]; cb[idx]=c_list[i]
            zb,_ = solve_qubo_bruteforce(Qb, cb)
            zb = np.asarray(zb,int).ravel()
            for k,i in enumerate(units):
                zy,zu,zv = map(int, zb[3*k:3*k+3])
                old = local_augL(y[i,t],u[i,t],v[i,t],␣
 ↪s_y[i,t],s_u[i,t],s_v[i,t],
                                 lam_y[i,t],lam_u[i,t],lam_v[i,t],␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t])
                new = local_augL(y[i,t],u[i,t],v[i,t],␣
 ↪s_y[i,t],s_u[i,t],s_v[i,t],
                                 lam_y[i,t],lam_u[i,t],lam_v[i,t], zy,zu,zv)
```

```
                if new + ACCEPT_TOL <= old:
                    state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪zy,zu,zv
                else:
                    state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t]

def block2_batched_dvqe(state):
    if DVQE is None: raise RuntimeError("DVQE not available")
    y,u,v, s_y,s_u,s_v = state["y"],state["u"],state["v"],␣
 ↪state["s_y"],state["s_u"],state["s_v"]
    lam_y,lam_u,lam_v = state["lam_y"],state["lam_u"],state["lam_v"]
    z_y_prev, z_u_prev, z_v_prev = state["z_y"].copy(), state["z_u"].copy(),␣
 ↪state["z_v"].copy()
    for t in range(T):
        qy = -(lam_y[:,t] + rho_y*(y[:,t] + s_y[:,t])) + 0.5*rho_y
        qu = -(lam_u[:,t] + rho_u*(u[:,t] + s_u[:,t])) + 0.5*rho_u
        qv = -(lam_v[:,t] + rho_v*(v[:,t] + s_v[:,t])) + 0.5*rho_v
        Q_list=[]; c_list=[]
        for i in range(N):
            Q3,c3 = micro_qubo_coeffs(qy[i], qu[i], qv[i], y[i,t], u[i,t],␣
 ↪v[i,t],
                                      (z_y_prev[i,t-1] if t>0 else␣
 ↪state["y0"][i]))
            Q_list.append(Q3); c_list.append(c3)
        groups = batch_partition(Q_list, c_list, k=K_BATCHES)
        for units in groups:
            m=len(units); Qb=np.zeros((3*m,3*m)); cb=np.zeros(3*m)
            for k,i in enumerate(units):
                idx=slice(3*k,3*k+3); Qb[idx,idx]=Q_list[i]; cb[idx]=c_list[i]
            zb,_,_ = DVQE(mode=dvqe_mode, Q=Qb, q_linear=cb, init_type=2,
                          depth=dvqe_depth, lr=dvqe_lr,␣
 ↪max_iters=dvqe_max_iters,
                          qpu_qubit_config=qpu_qubit_config_batch, rel_tol=1e-6)
            zb = np.asarray(zb,int).ravel()
            for k,i in enumerate(units):
                zy,zu,zv = map(int, zb[3*k:3*k+3])
                old = local_augL(y[i,t],u[i,t],v[i,t],␣
 ↪s_y[i,t],s_u[i,t],s_v[i,t],
                                 lam_y[i,t],lam_u[i,t],lam_v[i,t],␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t])
                new = local_augL(y[i,t],u[i,t],v[i,t],␣
 ↪s_y[i,t],s_u[i,t],s_v[i,t],
                                 lam_y[i,t],lam_u[i,t],lam_v[i,t], zy,zu,zv)
                if new + ACCEPT_TOL <= old:
```

```
                    state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪zy,zu,zv
                else:
                    state["z_y"][i,t], state["z_u"][i,t], state["z_v"][i,t] =␣
 ↪z_y_prev[i,t],z_u_prev[i,t],z_v_prev[i,t]

# =============== ADMM driver ===============
def run_variant(block2_func, label, print_every=10):
    state = reset_state(seed=7)
    residuals = []
    for it in range(max_iter):
        block1_qp(state)
        block2_func(state)
        block3_and_duals(state)
        res = residual(state)
        residuals.append(res)
        if it == 0:
            print(f"[{label}] iter={it:4d}  residual={res:.4e}")
        # print after each 10 iterations: 10, 20, 30, ...
        if (it + 1) % print_every == 0:
            print(f"[{label}] iter={it+1:4d}  residual={res:.4e}")

        if res < epsilon:
            break

    # always print a final line
    print(f"[{label}] done at iter={len(residuals):4d} ␣
 ↪final_residual={residuals[-1]:.4e}")
    return label, np.array(residuals)


# =============== Helper to run a single variant ===============
def run_and_store(fn, label, curves):
    try:
        lbl, res = run_variant(fn, label)
        curves[lbl] = res
        print(f"{lbl}: iters={len(res)}, final={res[-1]:.3e}")
        return True
    except Exception as e:
        print(f"{label}: ERROR -> {e}")
        return False

# =============== Helper to plot a pair ===============
def plot_pair(title, brute_lbl, dvqe_lbl, outname, curves):
    if brute_lbl not in curves or dvqe_lbl not in curves:
        print(f"Skip {title}: missing results for one or both variants.")
        return
```

```
    res_b = curves[brute_lbl]
    res_d = curves[dvqe_lbl]

    plt.figure(figsize=(7.2, 4.6))
    plt.semilogy(res_b, linestyle="-",  marker="o", linewidth=2.3,␣
 ↪label=brute_lbl)
    plt.semilogy(res_d, linestyle="--", marker="s", linewidth=2.3,␣
 ↪label=dvqe_lbl)

    plt.grid(True, which="both", linestyle="--", linewidth=0.6)
    plt.xlabel("ADMM iteration", fontsize=14)
    plt.ylabel("Primal residual", fontsize=14)
    plt.title(title, fontsize=15)

    leg = plt.legend(loc="best", frameon=True, fontsize=12)
    leg.get_frame().set_alpha(0.85)

    ax = plt.gca()
    ax.spines["right"].set_visible(False)
    ax.spines["top"].set_visible(False)
    plt.xticks(fontsize=12); plt.yticks(fontsize=12)

    plt.tight_layout()
    plt.savefig(outname, dpi=600, bbox_inches="tight")  # save as jpg
    plt.close()
    print(f"Saved: {outname}")

# =============== Define pairs (title, [(label, fn), (label, fn)], outname)␣
 ↪===============
pairs = [
    ("3-QUBO: Brute Force vs DVQE",
     [("3 (Brute Force)",       block2_3qubo_bruteforce),
      ("3 (DVQE)",              block2_3qubo_dvqe)],
     "3qubo_vs_dvqe.jpg"),

    ("Micro QUBO: Brute Force vs DVQE",
     [("Micro (Brute Force)",   block2_micro_bruteforce),
      ("Micro (DVQE)",          block2_micro_dvqe)],
     "micro_vs_dvqe.jpg"),

    ("Batched QUBO: Brute Force vs DVQE",
     [("Batched (Brute Force)", block2_batched_bruteforce),
      ("Batched (DVQE)",        block2_batched_dvqe)],
     "batched_vs_dvqe.jpg"),
]
```

```python
# =============== Run pair-by-pair ===============
curves = {}
for title, variants2, outname in pairs:
    print(f"\n=== Running pair: {title} ===")
    ok = True
    for label, fn in variants2:
        ok = run_and_store(fn, label, curves) and ok

    brute_lbl, dvqe_lbl = variants2[0][0], variants2[1][0]
    if ok:
        plot_pair(title, brute_lbl, dvqe_lbl, outname, curves)
    else:
        print(f"Skipped plot for {title} due to errors.")
```

```
=== Running pair: 3-QUBO: Brute Force vs DVQE ===
[3 (Brute Force)] iter=   0  residual=2.2790e+00
[3 (Brute Force)] iter=  10  residual=8.0191e-01
[3 (Brute Force)] iter=  20  residual=1.3012e-01
[3 (Brute Force)] iter=  30  residual=1.4621e-01
[3 (Brute Force)] iter=  40  residual=9.1694e-02
[3 (Brute Force)] iter=  50  residual=9.1462e-02
[3 (Brute Force)] iter=  60  residual=8.4856e-02
[3 (Brute Force)] iter=  70  residual=1.1058e-01
[3 (Brute Force)] iter=  80  residual=1.1340e-01
[3 (Brute Force)] iter=  90  residual=1.3980e-01
[3 (Brute Force)] iter= 100  residual=1.0687e-01
[3 (Brute Force)] iter= 110  residual=9.1808e-02
[3 (Brute Force)] iter= 120  residual=8.1900e-01
[3 (Brute Force)] iter= 130  residual=1.0623e-01
[3 (Brute Force)] iter= 140  residual=8.3401e-01
[3 (Brute Force)] iter= 150  residual=7.7797e-02
[3 (Brute Force)] iter= 160  residual=1.4409e-02
[3 (Brute Force)] iter= 170  residual=2.6688e-03
[3 (Brute Force)] done at iter= 176  final_residual=9.7036e-04
3 (Brute Force): iters=176, final=9.704e-04
[3 (DVQE)] iter=   0  residual=2.2790e+00
[3 (DVQE)] iter=  10  residual=1.0068e-01
[3 (DVQE)] iter=  20  residual=1.2883e-01
[3 (DVQE)] iter=  30  residual=1.4615e-01
[3 (DVQE)] iter=  40  residual=9.1646e-02
[3 (DVQE)] iter=  50  residual=9.1392e-02
[3 (DVQE)] iter=  60  residual=8.4893e-02
[3 (DVQE)] iter=  70  residual=1.1049e-01
[3 (DVQE)] iter=  80  residual=1.1339e-01
[3 (DVQE)] iter=  90  residual=1.3979e-01
[3 (DVQE)] iter= 100  residual=1.0688e-01
[3 (DVQE)] iter= 110  residual=9.1813e-02
```

```
[3 (DVQE)] iter= 120  residual=8.1904e-01
[3 (DVQE)] iter= 130  residual=1.0623e-01
[3 (DVQE)] iter= 140  residual=8.3407e-01
[3 (DVQE)] iter= 150  residual=7.7809e-02
[3 (DVQE)] iter= 160  residual=1.4412e-02
[3 (DVQE)] iter= 170  residual=2.6693e-03
[3 (DVQE)] done at iter= 176  final_residual=9.7052e-04
3 (DVQE): iters=176, final=9.705e-04
Saved: 3qubo_vs_dvqe.jpg

=== Running pair: Micro QUBO: Brute Force vs DVQE ===
[Micro (Brute Force)] iter=   0  residual=2.7037e+00
[Micro (Brute Force)] iter=  10  residual=1.2236e-01
[Micro (Brute Force)] iter=  20  residual=8.3481e-02
[Micro (Brute Force)] iter=  30  residual=8.1973e-02
[Micro (Brute Force)] iter=  40  residual=1.2310e-01
[Micro (Brute Force)] iter=  50  residual=1.6446e-02
[Micro (Brute Force)] iter=  60  residual=3.0501e-03
[Micro (Brute Force)] done at iter=  67  final_residual=9.3958e-04
Micro (Brute Force): iters=67, final=9.396e-04
[Micro (DVQE)] iter=   0  residual=2.7342e+00
[Micro (DVQE)] iter=  10  residual=1.1943e-01
[Micro (DVQE)] iter=  20  residual=8.3934e-02
[Micro (DVQE)] iter=  30  residual=8.9134e-01
[Micro (DVQE)] iter=  40  residual=9.7926e-02
[Micro (DVQE)] iter=  50  residual=9.6550e-03
[Micro (DVQE)] iter=  60  residual=1.7925e-03
[Micro (DVQE)] done at iter=  64  final_residual=9.1512e-04
Micro (DVQE): iters=64, final=9.151e-04
Saved: micro_vs_dvqe.jpg

=== Running pair: Batched QUBO: Brute Force vs DVQE ===
[Batched (Brute Force)] iter=   0  residual=2.7037e+00
[Batched (Brute Force)] iter=  10  residual=1.2236e-01
[Batched (Brute Force)] iter=  20  residual=8.3481e-02
[Batched (Brute Force)] iter=  30  residual=8.1973e-02
[Batched (Brute Force)] iter=  40  residual=1.2310e-01
[Batched (Brute Force)] iter=  50  residual=1.6446e-02
[Batched (Brute Force)] iter=  60  residual=3.0501e-03
[Batched (Brute Force)] done at iter=  67  final_residual=9.3958e-04
Batched (Brute Force): iters=67, final=9.396e-04
[Batched (DVQE)] iter=   0  residual=2.7037e+00
[Batched (DVQE)] iter=  10  residual=1.2236e-01
[Batched (DVQE)] iter=  20  residual=8.3481e-02
[Batched (DVQE)] iter=  30  residual=8.1973e-02
[Batched (DVQE)] iter=  40  residual=9.2169e-02
[Batched (DVQE)] iter=  50  residual=8.7799e-01
[Batched (DVQE)] iter=  60  residual=4.3114e-02
```

```
[Batched (DVQE)] iter=  70  residual=7.9856e-03
[Batched (DVQE)] iter=  80  residual=1.4792e-03
[Batched (DVQE)] done at iter=  83  final_residual=8.9200e-04
Batched (DVQE): iters=83, final=8.920e-04
Saved: batched_vs_dvqe.jpg
```

[ ]: