

Problem 1**1 Assumptions**

- Log-returns are truncated at 1σ , and jump variation term JV is set to 0.
- The following tranformations were applied, instead of conditions specified in the paper (Song et al, 2020, Page 11)
This modification makes it easy to implement, but the validity of this modifacaiton is not rigorously checked.
 - $\omega = \text{ReLU}(\omega')$
 - $\alpha = \text{Sigmoid}(\alpha')$
 - $\gamma = \text{Sigmoid}(\gamma')$
 - $\beta = \omega_L = \lambda = 0$ (Jump is truncated)
 - $\omega', \alpha', \gamma' \in \mathcal{R}$ is the parameter search space.
- Quasi-Log likelihood was modified to ensure the numerical stability.
It prevents oscillation at local optima, or diverging loss.
 - $\epsilon = 10^{-12}$
 - $L(V_{\text{realized}}, V_{\text{estimated}}) = -\sum \left(\log(V_{\text{estimated}}) + \frac{V_{\text{realized}}}{V_{\text{estimated}}} \right)$ when $V_{\text{estimated}} \geq \epsilon$
 - $L(V_{\text{realized}}, V_{\text{estimated}}) = -\sum \left(\frac{1}{\epsilon} \left(1 - \frac{V_{\text{realized}}}{\epsilon} \right) V_{\text{estimated}} + (\log(\epsilon) + \frac{V_{\text{realized}}}{\epsilon}) \right)$ when $V_{\text{estimated}} < \epsilon$
- Adagrad optimizer with learning rate 10^{-2} was applied with 50 epoch.
The 200 epoch is good for convergence, 50 epoch was selected to save time.
 - Due to slow convergence of realized GARCH-Ito model, last 50 days were used for MSPE comparison
- Equation (2.7) of Song et al(2020) was modified.
This modification reduces free varaiable ω_1, ω_2 into ω , but the validity of this modification is not regorously checkd.
 - Original equation : $\omega^g = \gamma(\rho_1 - \rho_2 + 2\rho_3)\omega_1 - (\rho - \gamma\rho_2 + 2\gamma\rho_3)\omega_2 + \dots$
 - Modified equation : $\omega^g = (\rho_1 - \rho_2 + 2\rho_3)(\gamma\omega_1 - \omega_2) + \dots$
where $\gamma\omega_1 - \omega_2 = \omega$

2 Results

- Daily GARCH model shows highest MSPE error.
- Unified GARCH-Ito and Realized GARCH-Ito model shows improvements over the daily model.
- Realized GARCH-Ito have room for improvements if the optimization epoch is increased and jump variance is considered.

Model	RV estimator	Optimization parameter	MSPE Error(full)	MSPE Error(last 50 days)
GARCH	PRV	α, γ	5.24e-12	7.43e-12
Unified GARCH-Ito	PRV	ω, α, γ	3.20e-12	3.30e-12
Realized GARCH-Ito	PRV	$\omega, \alpha, \gamma, \nu$	5.25e-07	2.68e-12

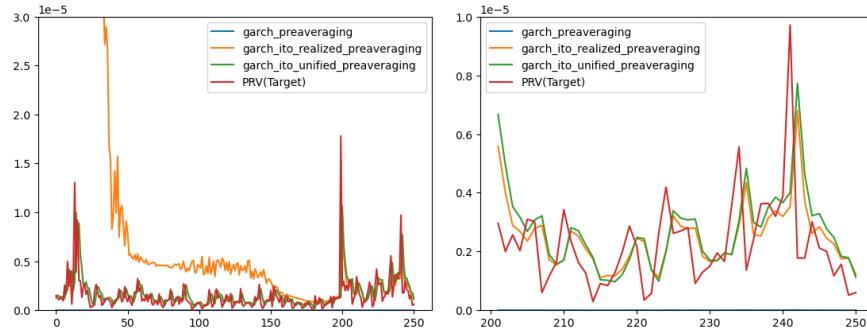


Figure 1: estimated volatility (left : 250 days, right: 50 days)

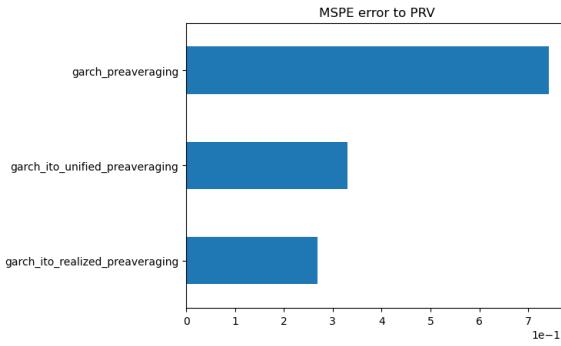


Figure 2: Last 50 day MSPE comparison

3 Code

Code can be accessed in https://github.com/Thessal/garch_ito

Optimizer

```

import numpy as np
import pandas as pd
import torch
from data import get_data
from vol_est import vol_est_arr

def likelihood(vol_est_arr, rv_arr, backend=torch):
    # Likelihood function for estimated == realized
    l11 = backend.log(vol_est_arr) + rv_arr / vol_est_arr
    # prevent divergence in vol < epsilon
    epsilon = 1e-12
    # l12 = (1 / epsilon - rv_arr / epsilon / epsilon) * vol_est_arr + (1 + rv_arr / epsilon)
    l12 = (1 / epsilon - rv_arr / epsilon / epsilon) * vol_est_arr + (float(np.log(epsilon)) + rv_arr / epsilon)
    l1 = backend.where(vol_est_arr>epsilon, l11, l12)
    return -backend.sum(l1)

def optimize_2param(rv_arr, init_params=(1e-3, 1e-3, 0.0), iter=1000, device=torch.device("cpu")):
    # Optimize gamma and beta. set omega = 0
    backend=torch
    vol_coef, rv_coef, resid = [torch.tensor(x, requires_grad=True) for x in init_params]
    resid = torch.tensor(0., requires_grad=True)
    # opt = torch.optim.Adam([vol_coef, rv_coef, resid], lr=1e-3)
    opt = torch.optim.Adam([vol_coef, rv_coef], lr=1e-2)
    for epoch in range(iter):
        opt.zero_grad()
        historical_est, next_est = vol_est_arr(rv_arr, vol_coef, rv_coef, backend=backend, device=device) # It looks like a RNN
        loss = -likelihood(historical_est, rv_arr, backend=backend)
        loss.backward()
        opt.step()
        #print(f"[{epoch}] loss:{loss.item():.2e}, params:{vol_coef.item():.2e}, {rv_coef.item():.2e}, {resid.item():.2e}, est:{next_est.item():.2e}")
    params = (vol_coef.item(), rv_coef.item(), resid.item())
    params_log = {"gamma": vol_coef.item(), "beta": rv_coef.item(), "omega_g": resid.item()}
    return params, params_log, next_est.item(), loss.item()

def optimize_3param(rv_arr, init_params=(0.5, 0.5, 1e-3), iter=1000, device=torch.device("cpu")):
    # Optimize gamma, beta, and omega
    backend=torch
    _omega, _beta, _gamma = [torch.tensor(x, requires_grad=True) for x in init_params]
    opt = torch.optim.Adagrad([\_omega, \_beta, \_gamma], lr=1e-2)

```

```

for epoch in range(iter):
    opt.zero_grad()
    gamma = torch.sigmoid(-gamma)
    beta = torch.sigmoid(-beta)
    omega = torch.relu(-omega)
    vol_coef = gamma
    rv_coef = (gamma-1)/beta*(torch.exp(beta)-1-beta)+torch.exp(beta)-1
    resid = (torch.exp(beta)-1)*torch.square(omega)/beta
    historical_est, next_est = vol_est_arr(rv_arr, vol_coef, rv_coef, resid, backend=backend, device=device) # It looks like a RNN
    loss = -likelihood(historical_est, rv_arr, backend=backend)
    loss.backward()
    opt.step()
    # print(f"\n[epoch] loss:{loss.item():.2e}, params:{vol_coef.item():.2e}, {rv_coef.item():.2e}, {resid.item():.2e}, est:{next_est.item():.2e}")
params = (-omega.item(), -beta.item(), -gamma.item())
params_log = {"gamma":vol_coef.item(), "beta_g":rv_coef.item(), "omega_g":resid.item()}
return params, params_log, next_est.item(), loss.item()

def optimize_4param(rv_arr, init_params=(0.1,0,0,0), iter=1000, device=torch.device("cpu")):
    # Optimize omega, alpha, gamma, nu
    backend=torch
    # https://arxiv.org/pdf/1907.01175
    omega, alpha, gamma, nu = [torch.tensor(x, requires_grad=True) for x in init_params]
    opt = torch.optim.Adamgrad([omega, alpha, gamma, nu], lr=1e-2)
    for epoch in range(iter):
        opt.zero_grad()
        # Page 7
        omega = torch.relu(-omega)
        alpha = torch.sigmoid(-alpha)
        gamma = torch.sigmoid(-gamma)
        nu = torch.sigmoid(-nu)

        exp_alpha = torch.exp(alpha)
        rho1 = (exp_alpha - 1.) / alpha
        rho2 = (exp_alpha - 1. - alpha) / alpha / alpha
        rho3 = (exp_alpha - 1. - alpha - alpha * alpha * 0.5) / (alpha*alpha*alpha)

        alpha_g = (rho1 - rho2 + 2. * gamma * rho3) * alpha
        # beta_g = 0
        og1 = (rho1 - rho2 + 2. * rho3) * omega # omega = gamma * omega1 - omega2
        og2 = (1. - gamma) * (rho2 - 2. * rho3) * nu
        omega_g = og1 + og2

        vol_coef = gamma
        rv_coef = alpha_g
        resid = omega_g
        historical_est, next_est = vol_est_arr(rv_arr, vol_coef, rv_coef, resid, backend=backend, device=device) # It looks like a RNN
        loss = -likelihood(historical_est, rv_arr, backend=backend)
        loss.backward()
        opt.step()
        # print(f"\n[epoch] loss:{loss.item():.2e}, params:{vol_coef.item():.2e}, {rv_coef.item():.2e}, {resid.item():.2e}, est:{next_est.item():.2e}")
params = (-omega.item(), -alpha.item(), -gamma.item(), -nu.item())
params_log = {"gamma":vol_coef.item(), "beta_g":rv_coef.item(), "omega_g":resid.item()}
return params, params_log, next_est.item(), loss.item()

def loop(rv_estimate_fn, optimize_fn, save_name, initial_param, iter=200):
    backend = torch
    device = torch.device("cpu")
    df = get_data().loc["2023-11":"2025-11"]
    rv_arr_all = rv_estimate_fn(df, backend=backend, device=device)
    est_rv = dict()
    lookback = 500
    params = initial_param
    for i in range(lookback, len(rv_arr_all)):
        rv_arr = rv_arr_all[i-lookback:i]
        params, params_log, vol_pred, loss = optimize_fn(rv_arr, init_params=params, device=device, iter=iter)
        est_rv[i] = params_log
        est_rv[i].update({"vol_pred":vol_pred, "vol_real":rv_arr_all[i].item(), "loss":loss})
        print(" ".join([f"{k}:{v:.2e}" for k,v in est_rv[i].items()]))
        if i%10 == 0:
            result = pd.DataFrame(est_rv).T
            result.to_csv(f"result_{save_name}.csv")
    result = pd.DataFrame(est_rv).T
    result.to_csv(f"result_{save_name}.csv")

```

GARCH

```

import torch
import numpy as np
# GARCH model

def vol_est(vol_prev, rv, vol_coef, rv_coef, resid=0):
    # GARCH(1,1) information process
    # volatility at t := expectation of return^2(t) at (t-1)
    return vol_coef*vol_prev + rv_coef*rv + resid

def vol_est_arr(rv_arr, vol_coef, rv_coef, resid, backend=torch, device="cpu"):
    # Chained estimation of GARCH(1,1)
    # backend can be torch or np
    result = [torch.Tensor([0]).to(device)]
    for i in range(len(rv_arr)):
        result.append(vol_est(result[i], rv_arr[i], vol_coef, rv_coef, resid))
    historical = backend.concat(result[:-1])
    historical = backend.maximum(historical, torch.tensor(1e-6).to(device)) # for stability
    estimation = result[-1]
    return historical, estimation

```

RV estimation

```

import torch
import numpy as np

# High-frequency or Daily volatility calculation

def rv_daily(df, backend=torch, device="cpu"):
    # Daily realized volatility for standard GARCH(1,1) model
    x_d = df[["close"]].resample("1D").last()
    xi_d = np.log(x_d).diff()
    xi_d = xi_d - xi_d.mean()
    xi_sq = np.square(xi_d.values[1:])
    if backend==torch:
        return torch.Tensor(xi_sq).to(device)
    elif backend==np:
        return xi_sq

def rv_naive(df, backend=torch, device="cpu"):
    # RV estimation without pre-averaging
    logret = np.log(df[["close"]]).diff()
    logret = logret - logret.mean()
    logret = logret.resample("1D").transform(lambda x: np.clip(x-x.mean(), -x.std(), x.std()))
    rv = logret.pow(2).resample("1D").mean()
    # adjust to daily vol
    N = 60*60*24
    rv = rv.multiply(N).values
    if backend==torch:
        return torch.Tensor(rv).to(device)
    elif backend==np:
        return rv

def rv_preaveraged(df, backend=torch, device="cpu"):
    # Preaveraged RV
    logprec = np.log(df[["close"]])
    logret = logprec.diff()
    logret = logret.resample("1D").transform(lambda x: np.clip(x-x.mean(), -x.std(), x.std()))
    eta_hat = logret.pow(2).resample("1D").mean()**0.5
    N = 60*60*24
    K = int(np.sqrt(N))
    weight_function = np.minimum(np.linspace(0, 1, K), np.linspace(1, 0, K))
    def convolve(x,g):
        assert len(x) > len(g)
        result = np.convolve(x, weight_function, "same")
        result[:len(g)] = np.nan
        result[-len(g):] = np.nan
        return result
    logprec_avg = logprec.resample("1D").transform(lambda x: convolve(x, weight_function))
    ybar = logprec_avg.diff()
    zeta = 1 / K
    psi = 1/12
    prv = ((ybar.pow(2)).resample("1D").mean() - zeta * eta_hat)/psi
    # adjust to daily vol
    # prv = prv.multiply(float(N)).values
    prv = prv.values
    if backend==torch:
        return torch.Tensor(prv).to(device)
    elif backend==np:
        return prv

```