Control Variates for GARCH Model

Ludovic Guyot - Mourad Chikhi - Michael Aichoun

ENSAE Paris

April, 2023

Table of Contents

Introduction

- Overview of MCMC principle
- Theoretical explanation of Variance reduction in MCMC
- Application of variance reduction in MCMC to GARCH model

Markov Chain Monte Carlo (MCMC)

- MCMC is a statistical method for sampling from complex probability distributions
- It generates a Markov chain of the parameters that converges to the target distribution
- Key idea: construct a Markov chain where the states are samples from the target distribution
- Typical steps of an MCMC algorithm include proposing a new state, evaluating the density, f, of the new state relatively to the previous, and updating the chain if it accepts

Control Variates - ZV-MCMC - in general

- General expressions for control variates in ZV method
- Using Schrödinger-type Hamiltonian and trial function

$$\psi(x) = \frac{P(x)}{\sqrt{\pi(x)}}$$

$P(x)$ assumed to be a polynomial and π the a posterior distribution

- Renormalized function:

$$\tilde{f(x)} = f(x) - \frac{1}{2}\Delta P(x) + \nabla P(x) \cdot z$$

- Where:

$$z = -\frac{1}{2}\nabla \ln \pi(x)$$

$$\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d}\right) \text{ denotes the gradient}$$

$$\Delta = \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2}$$

Control Variates - ZV-MCMC - with polynomial of degree 1

- 1st degree polynomial specification: $P(x) = \sum_{j=1}^{d} a_j x_j$
- Renormalized function becomes:

$$\tilde{f(x)} = f(x) + \frac{H\psi(x)}{\sqrt{\pi(x)}} = f(x) + a^T z$$

- Optimal a minimizes the variance of $\tilde{f(x)}$:

$$a = -\Sigma_{zz}^{-1}\sigma(z, f)$$

- Where:

$$\Sigma_{zz} = \text{cov}(z, z^T)$$
$$\sigma(z, f) = \text{cov}(z, f)$$

It means that $-a$ is the estimated coefficient from a OLS regression of f on z

GARCH(1,1) Model

The conditional volatility is defined by the following equation:

$$h_t = \omega_1 + \omega_2 r_{t-1}^2 + \omega_3 h_{t-1}$$

Where:

- h_t is the conditional volatility at time t
- ω_1 is the coefficient of the conditional volatility intercept term
- ω_2 is the coefficient of the AutoRegressive Conditional Heteroskedasticity (ARCH) term of order 1
- ω_3 is the coefficient of the Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) term of order 1
- h_{t-1} is the conditional volatility at the previous time step
- r_{t-1} is the previous value of the GARCH process

The GARCH process is defined by the following equation:

$$r_t = \epsilon_t \sqrt{h_t}$$

where ϵ_t is a weak white noise.

Posterior function of GARCH(1,1)

$r_t|F_{t-1}$, the garch process, is supposed to follow a normal law distribution $(0, h_t)$ and the unormalized likelyhood function is thus:

$$L(\omega_1, \omega_2, \omega_3|r) = \prod_{t=1}^{T} \frac{1}{\sqrt{h_t}} \exp\left(-\frac{r_t^2}{2h_t}\right)$$

and using independent truncated Normal priors for the parameters, the posterior is:

$$\pi(\omega_1, \omega_2, \omega_3|r) \propto \exp[-\frac{1}{2}(\frac{\omega_1}{\sigma^2(\omega_1)} + \frac{\omega_2}{\sigma^2(\omega_2)} + \frac{\omega_3}{\sigma^2(\omega_3)})] \prod_{t=1}^{T} \frac{1}{\sqrt{h_t}} \exp\left(-\frac{r_t^2}{2h_t}\right)$$

Control Variates for First Degree Polynomial for GARCH process

The control variates are given by the derivative of the posterior wrt each parameter:

$$\frac{\partial \ln \pi}{\partial \omega_i} = -\frac{\omega_i}{\sigma^2(\omega_i)} - \frac{1}{2}\sum_{t=1}^{T}\frac{1}{h_t}\frac{\partial h_t}{\partial \omega_i} + \frac{1}{2}\sum_{t=1}^{T}\frac{r_t^2}{h_t^2}\frac{\partial h_t}{\partial \omega_i}$$

where $i = 1, 2, 3$, and

$$\frac{\partial h_t}{\partial \omega_1} = \frac{1-\omega_3^{t-1}}{1-\omega_3}.$$

$$\frac{\partial h_t}{\partial \omega_2} = \left(r_{t-1}^2 + \omega_3 \frac{\partial h_{t-1}}{\partial \omega_2}\right) 1(t > 1),$$

$$\frac{\partial h_t}{\partial \omega_3} = \left(h_{t-1} + \omega_3 \frac{\partial h_{t-1}}{\partial \omega_3}\right) 1(t > 1),$$

Python Program and results

- Presentation of the Metropolis-hasting algorythm for GARCH parameters estimation (Implementation steps)
- Presentation of Zero Variance MCMC technique for variance reduction with different control variates:
  - First degree polynomial
  - Second degree polynomial
- Amelioration with:
  - Lasso regression
  - Subsampling techniques

Metropolis-Hastings Sampler

Overview:

- Metropolis-Hastings is a Markov chain Monte Carlo (MCMC) method used to sample from a target distribution.
- It is particularly useful when direct sampling is not possible or efficient.
- The algorithm is based on a random walk proposal and an acceptance/rejection step.

Notations:

- Target distribution: $L(\omega_1, \omega_2, \omega_3 | X)$, where X is the data.
- Proposal distribution: $q(\omega_1', \omega_2', \omega_3' | \omega_1, \omega_2, \omega_3)$, for generating a proposed sample.
- Current sample: $\omega^{(t)} = (\omega_1^{(t)}, \omega_2^{(t)}, \omega_3^{(t)})$.
- Proposed sample: $\omega'^{(t)} = (\omega_1'^{(t)}, \omega_2'^{(t)}, \omega_3'^{(t)})$.
- Acceptance ratio: $a = \frac{L(X|\omega_1'^{(t)},\omega_2'^{(t)},\omega_3'^{(t)}) \cdot q(\omega_1^{(t)},\omega_2^{(t)},\omega_3^{(t)}|\omega_1'^{(t)},\omega_2'^{(t)},\omega_3'^{(t)})}{L(X|\omega_1^{(t)},\omega_2^{(t)},\omega_3^{(t)}) \cdot q(\omega_1'^{(t)},\omega_2'^{(t)},\omega_3'^{(t)}|\omega_1^{(t)},\omega_2^{(t)},\omega_3^{(t)})}$.

Code for Posterior function

```
def g_i(r, h):
    return -0.5 * np.log(h) - 0.5 * (r**2 / h) # Normal law because r_{t}|F_t-1 follow N(0,h_t)

def g(x , data, h_initial):
    h = h_initial # max g_i initial
    LL = g_i(data[0],h) # density is initialized with our first simulated data

    for i in range(1, len(data)):
        h = x[0] + x[1] * data[i-1]**2 + x[2] * h
        LL += g_i(data[i],h)

    return LL
```

Figure: density function

Code for Metropolis-Hastings Sampler

```
while numaccept < M:

    Y = X + np.random.normal(loc=0, scale=sigma, size=3)  # Proposal move with a normal distribution (dim=3 because 3 parameters)
    U = np.log(np.random.rand(1)) # Random draw in a uniform distribution for accept/reject

    if np.all(Y > 0): # Reject all samples where any parameter is negative (truncated normal distribution)
        alpha = g(Y, data = r_garch, h_initial = h) - g(X, data = r_garch, h_initial = h)  # Difference between current and previous dens.
        if U < alpha:  # Reject if the ratio of densities is lower than a draw in a uniform distribution
            X = Y  # Accept proposal
            numaccept += 1 # Counts the number of acceptance in the accept/reject algorithm

            x1list.append(X[0])   # Add the values to the lists of omega
            x2list.append(X[1])   # Add the values to the lists of alpha
            x3list.append(X[2])   # Add the values to the lists of beta


    # Output progress report (can be long)
    i += 1 # Counts the number of total iteration
    if (i % 1000) == 0:
        print(" ...", i)
```

Figure: Metropolis algorithm

Simulated sample of parameters

We present result from a simulated GARCH(0.3,0.2,0.5). We also test our code on real return on exchange rate.

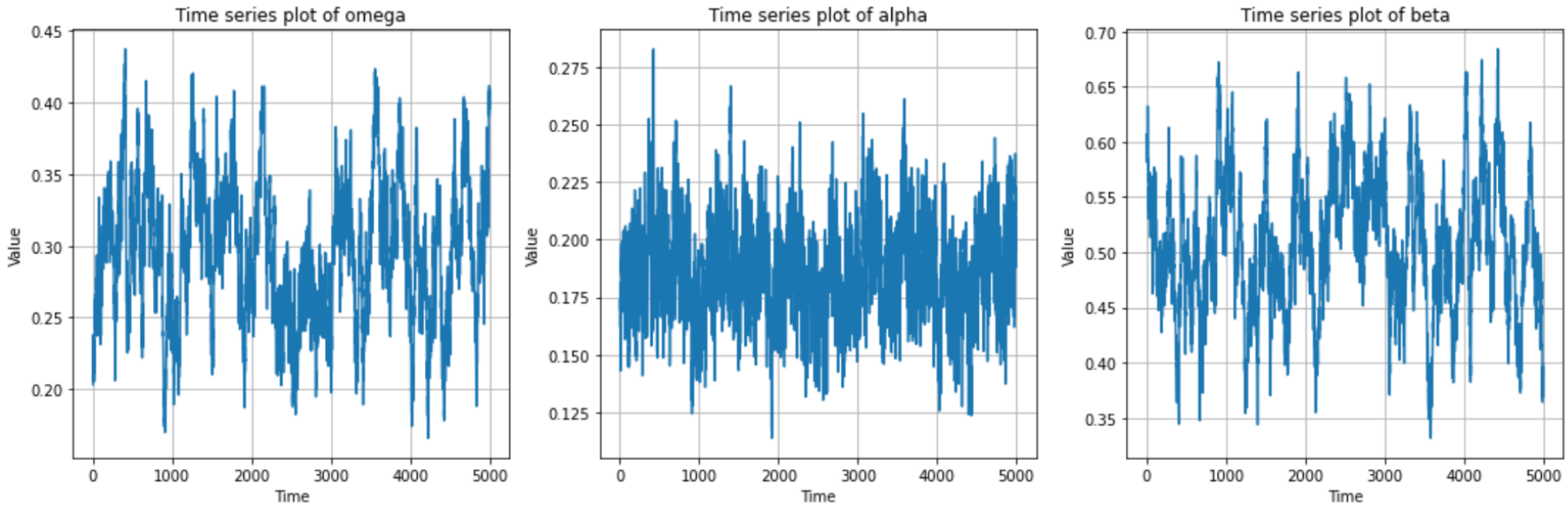


Figure: omega, alpha and beta values over chain from Metropolis algorythm

Simulated density of parameters

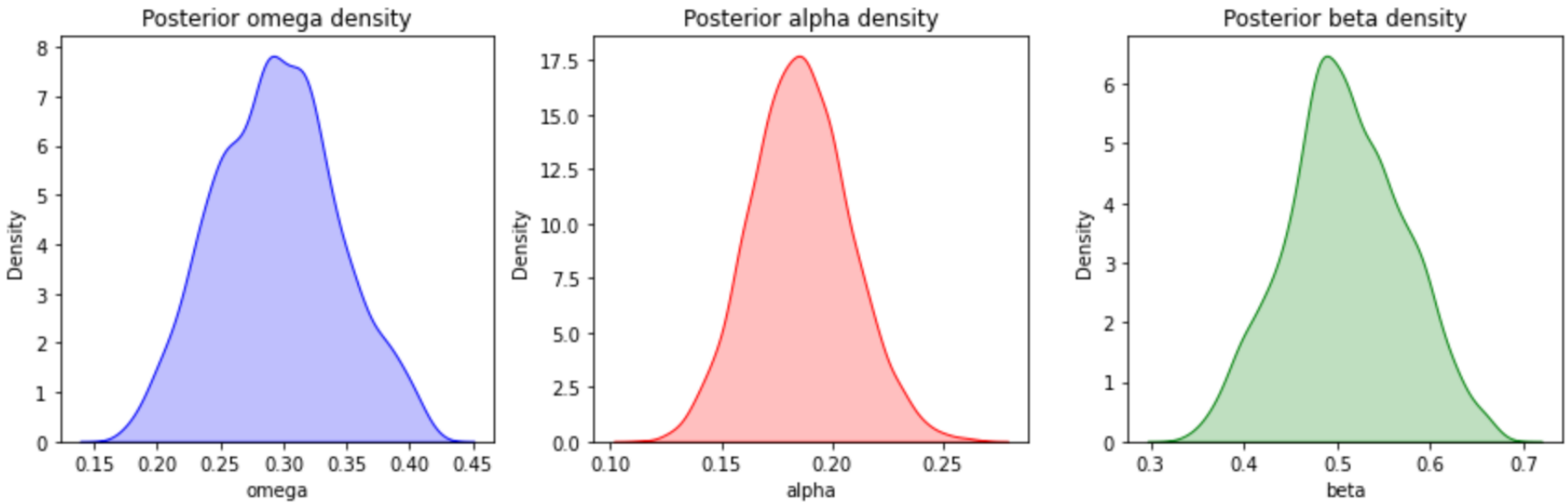


Figure: omega, alpha and beta density from Metropolis algorythm simulated chain

Zero Variance MCMC : 1st degree polynomials

Practical step of the ZV-MCMC 1st degree polynomial:

- Run a short Metropolis simulation
- Compute the three control variates, as well as the posterior, on each value of the short Metropolis simulation
- Regress the density on control variates (simple OLS regression)
- Create the new reduced variance function based on predicted residual of the regression
- Run a long Metropolis simulation with the new function

Control variates example

```python
def z2_i(r,h,r_lag,h_lag, x, i,h_prime):
    return  (h_prime) * (-(0.5/h) + (0.5 * r**2)/(h**2))

def z2(x , data, h_initial):
    Z = 0
    h_t = h_initial
    h_prime = 0 #initial derivatves of h wrt alpha (x[1])

    for i in range(1, len(data)):
        h = h_t
        h_prime = data[i-1]**2 + x[2]*h_prime
        h_t = x[0] + x[1] * data[i-1]**2 + x[2] * h
        Z += z2_i(data[i],h_t,data[i-1],h,x,i,h_prime)

    return -1/2 * (- x[1]/v_emp_alpha + Z)
```

Figure: Example of our control variates: second term of the gradient

Comparison with ZV-MCMC P1 with simple Metropolis

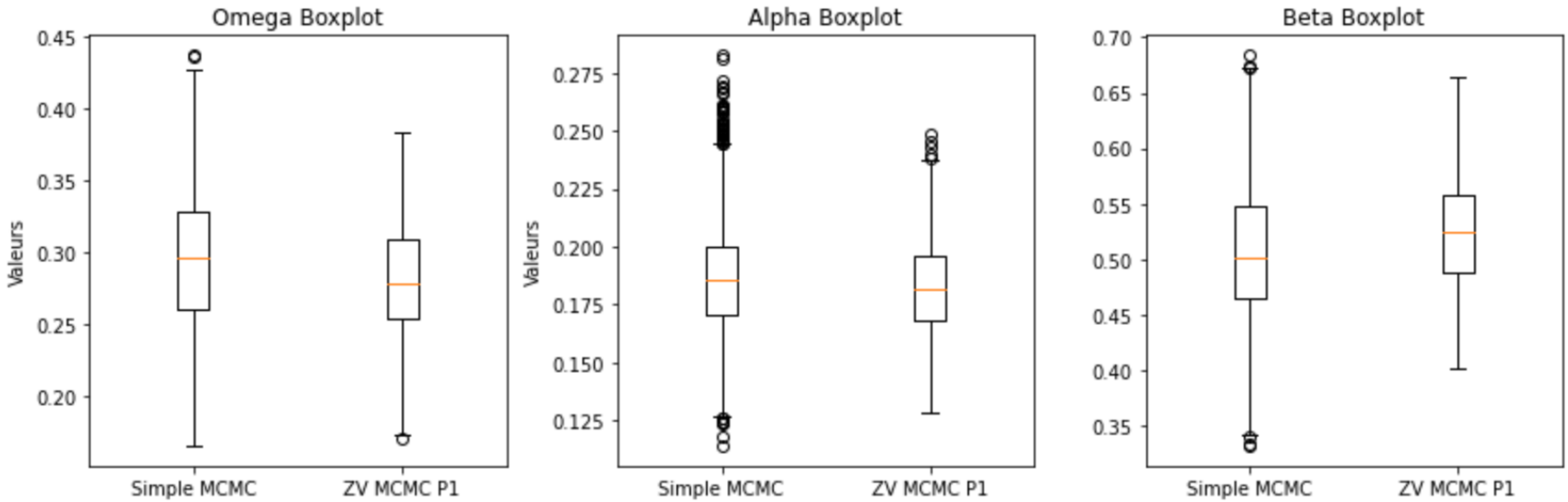


Figure: Box Plot of omega, alpha and beta for both method

Comparison with ZV-MCMC P1 with simple Metropolis

We present here another boxplot. We implement other control variate, which do not follow exactly the method from the paper. We indeed define control variate without lag on this version. We obtained better results with this other method, closer to what we wanted to have.

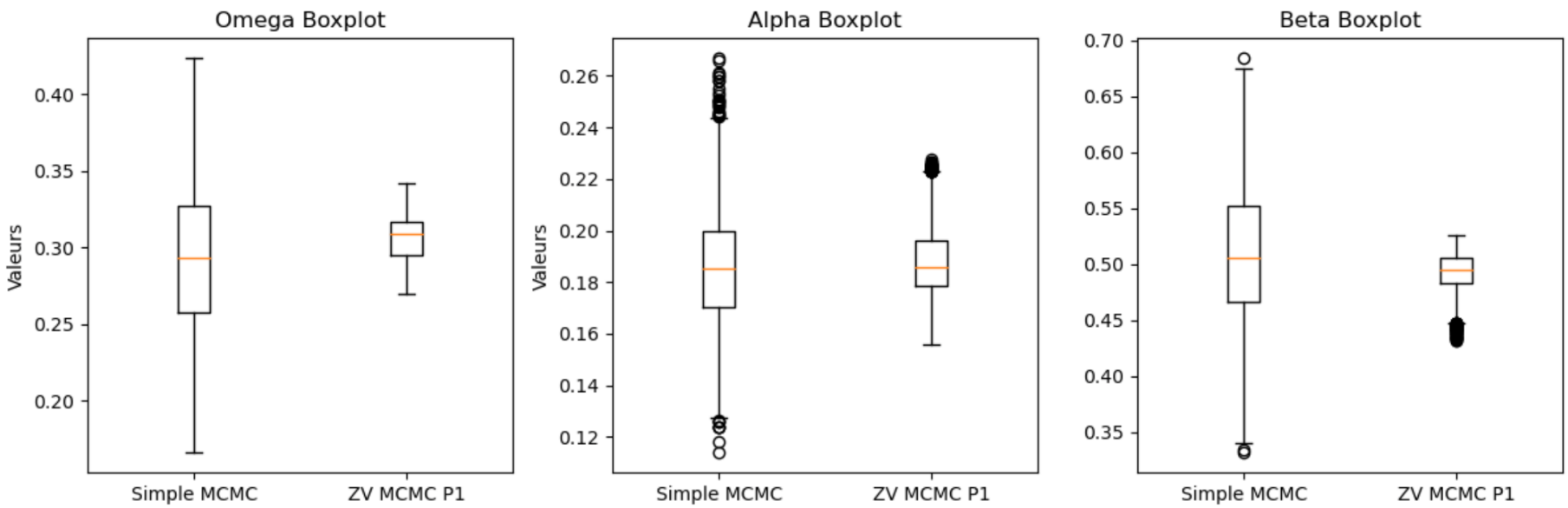


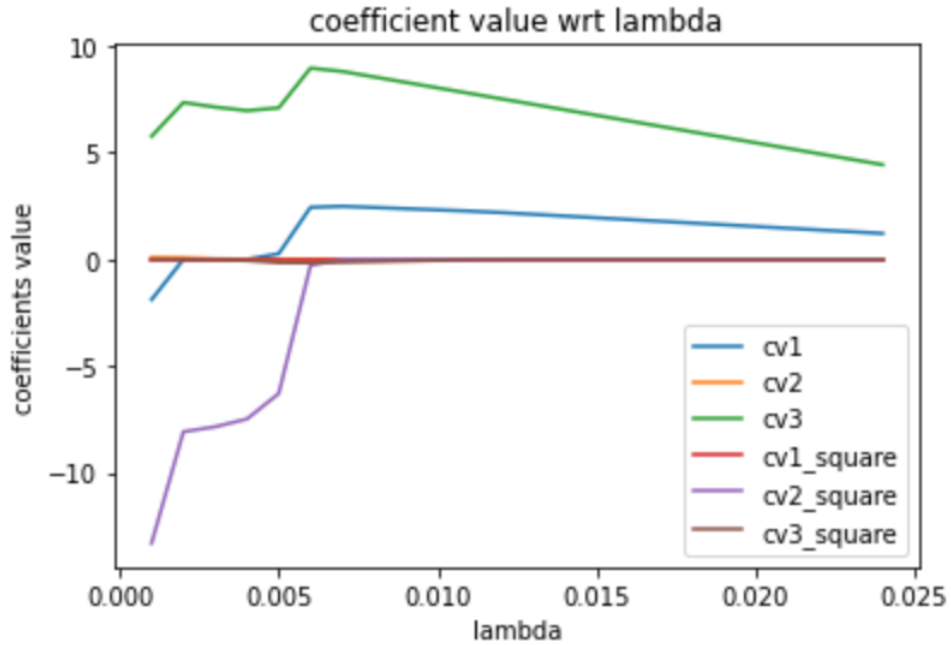
Figure: Box Plot of omega, alpha and beta for both method

Zero Variance MCMC : second degree polynomials

Since we did not obtain the desired results for the polynomial of degree 1, we decide to try another method for the polynomial of degree 2. We regress density on x and x^2 to approximate a polynomial of degree 2 with the folowing specification:

$$g(x) = a^t x + b^t x^2$$

And we incorporate a Lasso regression to find the best of the 6 coefficients.



Comparison with of ZV-MCMC P2 with simple Metropolis

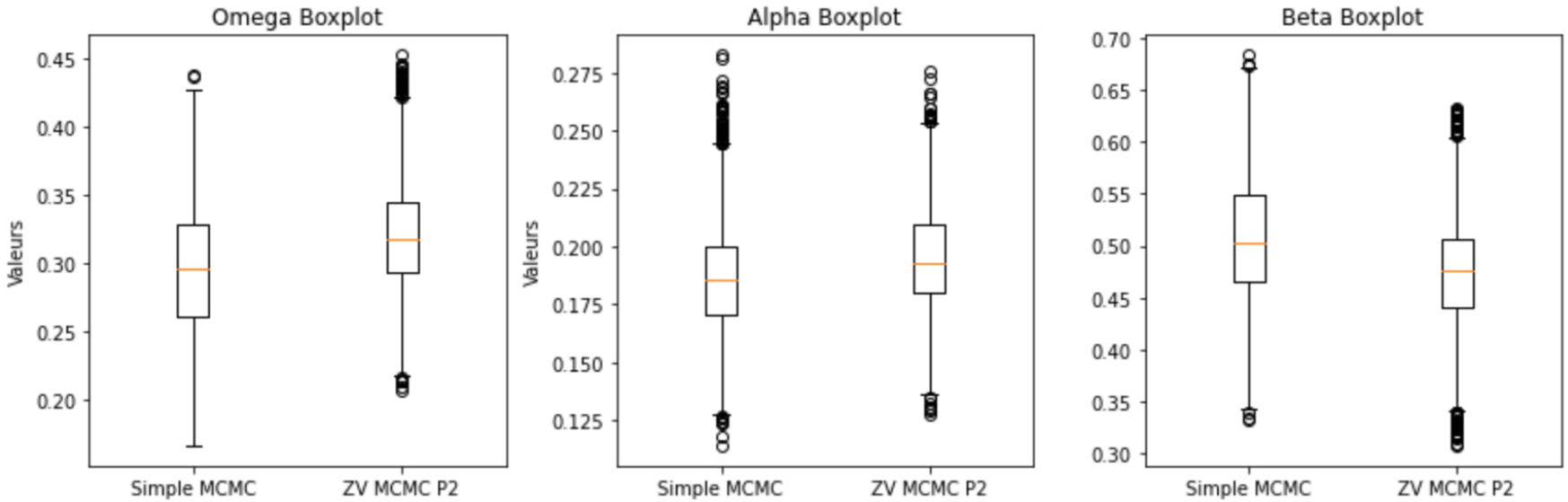


Figure: Box Plot of omega, alpha and beta for both method

Independance issue

OLS (and Lasso) regression relies in part on the assumption that the sample is independent and identically distributed.

- **Problem**: Metropolis sample is built from 3 Markov chains so the observations are necessarily dependent.
- **Solution**: To overcome this problem we would have to run the algorithm of step 1, i.e. question 1, a large number of times. Then, from the autocorrelation function, decide on the distance from which the observations are no longer correlated and choose a value among a number n according to the autocorrelation function.

This will reduce the autocorrelation problems in an efficient way.

Thank You!

Questions?

Bonus : Regression results

OLS Regression Results

Dep. Variable:	y	R-squared:	0.037
Model:	OLS	Adj. R-squared:	0.034
Method:	Least Squares	F-statistic:	8.378
Date:	Mon, 24 Apr 2023	Prob (F-statistic):	1.67e-05
Time:	13:26:05	Log-Likelihood:	-1688.3
No. Observations:	1000	AIC:	3385.
Df Residuals:	996	BIC:	3404.
Df Model:	3		
Covariance Type:	HC3		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1814.4513	0.180	-1.01e+04	0.000	-1814.803	-1814.099
x1	0.0064	0.002	3.707	0.000	0.003	0.010
x2	0.0022	0.001	1.942	0.052	-2e-05	0.004
x3	-0.0083	0.002	-4.388	0.000	-0.012	-0.005

Omnibus:	170.848	Durbin-Watson:	0.640
Prob(Omnibus):	0.000	Jarque-Bera (JB):	286.084
Skew:	-1.085	Prob(JB):	7.55e-63
Kurtosis:	4.470	Cond. No.	707.

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)

Figure: Regression result from the control variates coefficient of the first degree polynomial for the ZV-MCMC method

Bonus : different control variates which allow us to decrease variance of Metropolis simulation

```
def z2_i(r,h):
    return -0.5*(r**2)/h+0.5*(r**4)/(h**2)
def z2(x , data, h_initial):
    h = h_initial
    Z = z2_i(data[0],h)

    for i in range(1, len(data)):
        h = x[0] + x[1] * data[i-1]**2 + x[2] * h
        Z += z2_i(data[i],h)

    return Z
```

Figure: Example of control variates which allow to decrease variance of Metropolis simulation