1 The AR2 Model

Consider a mean zero AR(2) model conditioned on y_1 and y_2.

$$y_t = \phi_1 * y_{t-1} + \phi_2 * y_{t-2} + \epsilon_t \text{ for } t = 3, \ldots, T$$

,

where ϵ are iid $\mathcal{N}(0, \sigma^2)$.

Using the independent priors:

$$\begin{aligned} p(\phi_1) &\sim \mathcal{N}(p\bar{h}i_1, \tau_1^2) \\ p(\phi_2) &\sim \mathcal{N}(p\bar{h}i_2, \tau_2^2) \\ p(\sigma^2) &\sim \mathcal{IG}(\text{shape} = \alpha,\ \text{rate} = \beta), \end{aligned}$$

we have a model with zero stationary restrictions that can be easily sampled through Gibbs MCMC or approximated by a range of variational techniques.

There are four approaches considered:

- MCMC
- Mean field Variational Bayes
- Copula Variational Bayes
- A re-parameterised Copula Variational Bayes

2 Summary

Each approach uses the same model, $q(\theta) = BVN((\phi_1, \phi_2)'|\mu, \Sigma)IG(\sigma^2|\alpha, \beta)$ with estimated parameters

- μ_1: The mean of ϕ_1
- μ_2: The mean of ϕ_2
- L_{11}, L_{21}, L_{22}, the three components of the lower triangular decomposition of Σ
- α, β, the shape and rate parameters of σ^2.

The diagonal of L is not forced to be positive so the decomposition is not invariant to sign changes.

Collectively, we have $\lambda = (\mu_1, \mu_2, L_{11}, L_{21}, L_{22}, \alpha, \beta)'$.

```
> library(knitr)
> library(mvtnorm)
> library(reshape)
> summary = matrix( c(0.37, -0.30, 0.10, -0.03, 0.10, 49.2, 78.4, -77.5, 7.4,
```

```
> setwd("/home/nltom2/Desktop/Code/Approximate-Updating")
> library(ggplot2)
> library(GGally)
> thin <- read.csv("AR2thin.csv")[,-1]
> ggpairs(thin)
```

```
+                     0.37, -0.29, 0.09, 0, 0.10, 49.5, 79.8, -77.4, "1.3 x 10^{-6}",
+                     0.32, -0.32, 0.11, -0.01, 0.12, 46.4, 78.7, -77.2, 59.5,
+                     0.38, -0.30, -0.1, 0.03, -0.10, 48.7, 78.9, -77.4, 1.2) , byrow = TRUE
> colnames(summary) = c(paste0(expression(mu), 1), paste0(expression(mu), 2), "L11", "L21",
> rownames(summary) = c("Method of Moments", "Mean Field", "Copula 1", "Copula 2")
> kable(summary)
```

	mu1	mu2	L11	L21	L22	alpha	beta	L(theta)	time (seconds)*
Method of Moments	0.37	-0.3	0.1	-0.03	0.1	49.2	78.4	-77.5	7.4
Mean Field	0.37	-0.29	0.09	0	0.1	49.5	79.8	-77.4	1.3 x 10^{-6}
Copula 1	0.32	-0.32	0.11	-0.01	0.12	46.4	78.7	-77.2	59.5
Copula 2	0.38	-0.3	-0.1	0.03	-0.1	48.7	78.9	-77.4	1.2

Note that all of the variational algorithms are currently unparallelised, and the copula versions in particular have a huge potential for faster performance.

3 MCMC

Taking fifty thousand draws and discarding the first ten thousand, the effective sample sizes for each parameter is:

- ϕ_1 : 4500
- ϕ_2 : 4500
- σ^2 : 35000

Retaining only every tenth draws returns samples with effectively no autocorrelation. This thinned out sample is plotted below.

There is obvious eliptical dependence between ϕ_1 and ϕ_2, but not between either ϕ parameter and σ^2. The best fitting copula by BIC is gaussian, with the parameter $\rho = -0.29$.

```
> copest =data.frame(copula = c("gaussian", "t"), rho = c(-0.29, -0.29), df = c("NA", 112.5)
> kable(copest)
```

copula	rho	df	BIC
gaussian	-0.29	NA	-167.8
t	-0.29	112.5	-161.2

For consistency with the variational approaches, the method of moments is used to fit a parametric approximation of $q(\theta) = BVN((\phi_1, \phi_2)'|\mu, \Sigma)IG(\sigma^2|\alpha, \beta)$. This form is justified by the mean field approximation returning gaussian marginals for ϕ_1 and ϕ_2, and an inverse gamma marginal for σ^2. Combining this with the gaussian copula returns the bivariate normal distribution.

```
> MM = matrix( c(0.37, -0.30, 0.10, -0.03, 0.10, 49.2, 78.4, -77.5, 7.4), ncol = 9)
> colnames(MM) = c(paste0(expression(mu), 1), paste0(expression(mu), 2), "L11", "L21", "L22"
> kable(MM)
```

mu1	mu2	L11	L21	L22	alpha	beta	L(theta)	time (seconds)
0.37	-0.3	0.1	-0.03	0.1	49.2	78.4	-77.5	7.4

4 Mean Field Variational Bayes

The next step is to perform a mean-field variational bayes, which returns the optimal distribution families in a factorisable posterior approximaiton. These were found to be gaussian for ϕ_1 and ϕ_2, and inverse gamma for σ^2. The parameterisation for these are similar to the conditional distributions used in Gibbs sampling, with dependence on other parameter draws replaced with the corresponding expectations. The hyperparameters of these distributions depend on each other through the expectations and the algorithm iterates between these until convergence.

Result: Mean Field Approximation
Initialise λ randomly;
while *Not converged* **do**
  for $i = 1$ **to** 7 **do**
    Hold $\lambda_{j \neq i}$ fixed;
    Maximise λ_i ;
  end
end

Algorithm 1: Mean Field Variational Bayes

Mean Field Variational Bayes converged within five iterations, and the results are summarised below

```
> MFsum = matrix(c(0.37, -0.29, 0.09, 0, 0.10, 49.5, 79.8, -77.4, "1.3 x 10^{-6}"), ncol = 9
> colnames(MFsum) = c(paste0(expression(mu), 1), paste0(expression(mu), 2), "L11", "L21", "L
> kable(MFsum)
```

mu1	mu2	L11	L21	L22	alpha	beta	L(theta)	time (seconds)
0.37	-0.29	0.09	0	0.1	49.5	79.8	-77.4	1.3 x 10^{-6}

```
> library(pscl)
> library(gridExtra)
> mf <- read.csv("AR2mf.csv")[,-1]
> astar = 50
> xp1 = seq(-0.1, 1, length.out=1000)
> p1dens = dnorm(xp1, mf[10, 2], sqrt(mf[10, 3]))
> p1dat = data.frame(cbind(xp1, p1dens))
> p1plot = ggplot(p1dat, aes(x=xp1, y=p1dens)) + geom_line() + labs(x="Phi 1", y=NULL)
> xp2 = seq(-0.8, 0.4, length.out=1000)
> p2dens = dnorm(xp2, mf[10, 4], sqrt(mf[10, 5]))
> p2dat = data.frame(cbind(xp2, p2dens))
> p2plot = ggplot(p2dat, aes(x=xp2, y=p2dens)) + geom_line() + labs(x="Phi 2", y=NULL)
> xs = seq(0.5, 3.5, length.out = 1000)
> sdens = densigamma(xs, astar, mf[10, 1])
> sigdat = data.frame(cbind(xs, sdens))
> sigplot = ggplot(sigdat, aes(x=xs, y=sdens)) + geom_line() + labs(x="Sigma Squared", y=NUL
> grid.arrange(p1plot, p2plot, sigplot, ncol = 2)
```

5 Copula Variational Bayes 1

Remembering the Variational Bayes seeks to minimised the Evidence Lower Bound,

$$L(\theta, \lambda) = E_q \left[q_\theta (\log(p(\theta, y)) - \log(q(\theta))) \right]$$

We can take gradients as

$$\begin{aligned} \nabla_\lambda L(\theta, \lambda) &= E_q \left[\nabla_\lambda [q_\theta (\log(p(\theta, y)) - \log(q(\theta)))] \right] \\ &= E_q \left[\nabla_\lambda [\log(q_\theta)] (\log(p(\theta, y)) - \log(q(\theta))) \right] \\ &\approx 1/S \sum_{s=1}^{S} \nabla_\lambda [\log(q_{\theta^s})] (\log(p(\theta^s, y)) - \log(q(\theta^s))). \end{aligned} \tag{1}$$

This results in the Stochastic Gradient Ascent algorithm for Variational Bayes:

Result: Variational Approximation
Initialise λ to Methods of Moments Estimator;
while *Not converged* **do**
  Simulate θ_s for $s = 1, \dots S$ from $q(\theta^{t-1})$ **for** $i = 1$ **to** 7 **do**
    Hold $\lambda_{j \neq i}$ fixed;
    Calculate $\nabla^t_{\lambda_i} = 1/S \sum_{s=1}^{S} \nabla_\lambda [\log(q_{\theta^s})] (\log(p(\theta^s, y)) - \log(q(\theta^s)))$
  end
  Set $\lambda^t = \lambda^{t-1} + p_t \nabla^t_\lambda$ Set $t = t + 1$
end

Algorithm 2: Stochastic Gradient Ascent Algorithm 1

The Monte Carlo estimator has a huge variance and there were often draws of the same partial derivates between $-200,000$ and $200,000$ for α and β. Setting $S > 200$ was required, however the algorithm was extremely sensitive to initial conditions, so the Method of Moments estimators were used. Often α and β would take negative values so the natural log of these parameters was used instead.

Ranganath et al. 2014 reccomends setting

$$p_t = \eta diag(G_t)^{-1/2}$$

for some η, where G_t is the sum of the outer products of partial derivatives for $k = 1, \ldots, t$.

With $\eta = 0.1$ and $S = 200$, the algorithm took 244 iterations until the change in $L(\theta, \lambda)$ was less than 0.001.

6 Copula Variational Bayes 2

If we can transform $\theta = f(\epsilon, \lambda)$ where ϵ is parameter free, the derivative in (1) simplifies.

$$\begin{aligned} \nabla_\lambda L(\theta, \lambda) &= E_q \left[[q_\epsilon \nabla_\lambda [(\log(p(\theta, y)) - \log(q(\theta)))]] \right] \\ &\approx 1/S \sum_{s=1}^{S} \nabla_\lambda (\log(p(\theta^s, y)) - \log(q(\theta^s))) \end{aligned}$$

We can use $(\theta_1, \theta_2)' = \mu + Lc(\epsilon_1, \epsilon_2)'$ and $\theta_3 = Q^{-1}(\epsilon_3 | \alpha, \beta)$ where $(\epsilon_1, \epsilon_2) \sim N(0, I)$ and $\epsilon_3 \sim U(0, 1)$

Result: Variational Approximation
Initialise λ to Methods of Moments Estimator;
while *Not converged* **do**
  Simulate ϵ for $s = 1, \ldots S$ from $N(0, I)$ and $U(0, 1)$ Transform $\theta^s = f(\epsilon^s, \lambda^{t-1})$ **for** $i = 1$ **to** 7 **do**
    Hold $\lambda_{j \neq i}$ fixed;
    Calculate $\nabla^t_{\lambda_i} = 1/S \sum_{s=1}^{S} \nabla_\lambda [(\log(p(\theta^s, y)) - \log(q(\theta^s)))]$
  end
  Set $\lambda^t = \lambda^{t-1} + p_t \nabla^t_\lambda$ Set $t = t + 1$
end

Algorithm 3: Stochastic Gradient Ascent Algorithm 2

The resulting Monte Carlo draws from this algorithm have a much smaller variance and much less numerical instability than algorithm 1. S can be set to equal 1, massively speeding up the calculations. Further, it is unneccesary to transform α and β, and most parameters and invariant to their starting values. α and β are not invariant to their starting conditions, but after setting the other parameters to their converged values, it was found that $L(\theta, \lambda)$ is extremely flat for a wide range of α and β so the gradients are near zero.

The algorithm converged within a tolerance of 0.001 after 504 draws.

```
> SGA1 = matrix( c(0.32, -0.32, 0.11, -0.01, 0.12, 46.4, 78.7, -77.2, 59.5), ncol = 9)
> colnames(SGA1) = c(paste0(expression(mu), 1), paste0(expression(mu), 2), "L11", "L21", "L2
> kable(SGA1)
```

mu1	mu2	L11	L21	L22	alpha	beta	L(theta)	time (seconds)
0.32	-0.32	0.11	-0.01	0.12	46.4	78.7	-77.2	59.5

```
> L2 =  c(0.32, -0.32, 0.11, -0.01, 0.12, 46.4, 78.7)
> L = matrix(c(L2[3:4], 0, L2[5]), 2)
> Sigma = L %*% t(L)
> xp1 = seq(-0.1, 1, length.out=1000)
> p1dens = dnorm(xp1, L2[1], sqrt(Sigma[1, 1]))
> p1dat = data.frame(cbind(xp1, p1dens))
> p1plot = ggplot(p1dat, aes(x=xp1, y=p1dens)) + geom_line() + labs(x="Phi 1", y=NULL)
> xp2 = seq(-0.8, 0.4, length.out=1000)
> p2dens = dnorm(xp2, L2[2], sqrt(Sigma[2, 2]))
> p2dat = data.frame(cbind(xp2, p2dens))
> p2plot = ggplot(p2dat, aes(x=xp2, y=p2dens)) + geom_line() + labs(x="Phi 2", y=NULL)
> xs = seq(0.5, 3.5, length.out = 1000)
> sdens = densigamma(xs, L2[6], L2[7])
> sigdat = data.frame(cbind(xs, sdens))
> sigplot = ggplot(sigdat, aes(x=xs, y=sdens)) + geom_line() + labs(x="Sigma Squared", y=NUl
> x1 = seq(-1, 1, length.out=1000)
> x2 = seq(-1, 1, length.out=1000)
> z = matrix(0, length(x1), length(x2))
> for (i in 1:length(x1)) {
+   d1 = x1
+   d2= x2[i]
+   z[,i] = dmvnorm(cbind(d1,d2), L2[1:2], Sigma)
+ }
> colnames(z) = x1
> rownames(z) = x2
> # reshape the data
> dat.p1p2 <- melt(z)
> colnames(dat.p1p2) = c("Phi1", "Phi2", "value")
> p1p2 <- ggplot(dat.p1p2, aes(x=Phi1, y=Phi2, z = value)) + geom_contour()+ coord_equal()
> grid.arrange(p1plot, p2plot, sigplot, p1p2)
```

```
> SGA2 = matrix( c(0.38, -0.30, -0.1, 0.03, -0.10, 48.7, 78.9, -77.4, 1.2), ncol = 9)
> colnames(SGA2) = c(paste0(expression(mu), 1), paste0(expression(mu), 2), "L11", "L21", "L2
> kable(SGA2)
```

mu1	mu2	L11	L21	L22	alpha	beta	L(theta)	time (seconds)
0.38	-0.3	-0.1	0.03	-0.1	48.7	78.9	-77.4	1.2

```
> L2 =  c(0.38, -0.30, -0.1, 0.03, -0.10, 48.7, 78.9)
> L = matrix(c(L2[3:4], 0, L2[5]), 2)
> Sigma = L %*% t(L)
> xp1 = seq(-0.1, 1, length.out=1000)
> p1dens = dnorm(xp1, L2[1], sqrt(Sigma[1, 1]))
> p1dat = data.frame(cbind(xp1, p1dens))
> p1plot = ggplot(p1dat, aes(x=xp1, y=p1dens)) + geom_line() + labs(x="Phi 1", y=NULL)
> xp2 = seq(-0.8, 0.4, length.out=1000)
> p2dens = dnorm(xp2, L2[2], sqrt(Sigma[2, 2]))
> p2dat = data.frame(cbind(xp2, p2dens))
> p2plot = ggplot(p2dat, aes(x=xp2, y=p2dens)) + geom_line() + labs(x="Phi 2", y=NULL)
> xs = seq(0.5, 3.5, length.out = 1000)
> sdens = densigamma(xs, L2[6], L2[7])
> sigdat = data.frame(cbind(xs, sdens))
> sigplot = ggplot(sigdat, aes(x=xs, y=sdens)) + geom_line() + labs(x="Sigma Squared", y=NUl
> x1 = seq(-1, 1, length.out=1000)
> x2 = seq(-1, 1, length.out=1000)
> z = matrix(0, length(x1), length(x2))
> for (i in 1:length(x1)) {
+   d1 = x1
+   d2= x2[i]
+   z[,i] = dmvnorm(cbind(d1,d2), L2[1:2], Sigma)
+ }
> colnames(z) = x1
> rownames(z) = x2
> # reshape the data
> dat.p1p2 <- melt(z)
> colnames(dat.p1p2) = c("Phi1", "Phi2", "value")
> p1p2 <- ggplot(dat.p1p2, aes(x=Phi1, y=Phi2, z = value)) + geom_contour()+ coord_equal()
> grid.arrange(p1plot, p2plot, sigplot, p1p2)
```