# Autoregressive model of order $p$ (AR($p$))

## Parametrization

The autoregressive model of order $p$ (AR1($p$)) for the Gaussian vector $\mathbf{x} = (x_1, \ldots, x_n)$ is defined as (in obvious notation)

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \epsilon_t$$

for $t = p, \ldots, n$, and where the innovation process $\{\epsilon_t\}$ has fixed precision.

The AR($p$) process has an awkward parameterisation, as there are severe non-linear constraints on the $\phi$.-parameters for it to define a stationary model. Therefore we re-parameterized using the partial autocorrelation autocorrelation function, $\{\psi_k, k = 1, \ldots, p\}$, where $|\psi_k| < 1$ for all $k$ [1] and its *marginal (NOT conditional) precision* $\tau$. Furthermore, the joint distribution for $\{x_t, t = 1, \ldots, p\}$, is set to the stationary distribution for the process, hence there are no boundary issues.

## Hyperparameters

The marginal precision parameter $\tau$ is represented as

$$\theta_1 = \log(\tau)$$

and the prior for the marginal precision is defined on $\theta_1$. The partial autocorrelation function $\{\psi_k\}$ is represented

$$\psi_k = 2\frac{\exp(\theta_{k+1})}{1 + \exp(\theta_{k+1})} - 1$$

for $k = 1, \ldots, p$. The prior for $\{\theta_{k+1}, k = 1, \ldots, p\}$ is *defined* to be multivariate normal with mean $\mu$ and precision matrix $Q$.

## Specification

The AR($p$) model is specified inside the `f()` function as

```
f(<whatever>, model="ar", order=<p>, hyper = <hyper>)
```

The option `order` ($> 0$) is required. The multivariate normal prior for $\{\theta_{k+1}, k = 1, \ldots, p\}$, `is specified` as the parameters to the prior for $\theta_2$ (the first pacf-parameter), and the parameters to the multivariate normal prior (`mvnorm`), is `c(`$\mu, Q$`)`; see the example below.

### Hyperparameter spesification and default values

**hyper**

**theta1**

**hyperid** 15001
**name** log precision
**short.name** prec
**initial** 4
**fixed** FALSE
**prior** pc.prec
**param** 3 0.01
**to.theta** `function(x) log(x)`

[1] See for example `https://en.wikipedia.org/wiki/Partial_autocorrelation_function`. For $p = 1$, then $\psi_1 = \phi_1$, and for $p = 2$, then $\psi_1 = \phi_1/(1 - \phi_2)$ and $\psi_2 = \phi_2$.

**from.theta** `function(x) exp(x)`

**theta2**

**hyperid** 15002
**name** pacf1
**short.name** pacf1
**initial** 1
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.5
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta3**

**hyperid** 15003
**name** pacf2
**short.name** pacf2
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.4
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta4**

**hyperid** 15004
**name** pacf3
**short.name** pacf3
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.3
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta5**

**hyperid** 15005
**name** pacf4
**short.name** pacf4
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.2
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta6**

**hyperid** 15006
**name** pacf5

**short.name** pacf5
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta7**

**hyperid** 15007
**name** pacf6
**short.name** pacf6
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta8**

**hyperid** 15008
**name** pacf7
**short.name** pacf7
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta9**

**hyperid** 15009
**name** pacf8
**short.name** pacf8
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta10**

**hyperid** 15010
**name** pacf9
**short.name** pacf9
**initial** 0
**fixed** FALSE
**prior** pc.cor0

**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**theta11**
**hyperid** 15011
**name** pacf10
**short.name** pacf10
**initial** 0
**fixed** FALSE
**prior** pc.cor0
**param** 0.5 0.1
**to.theta** `function(x) log((1+x)/(1-x))`
**from.theta** `function(x) 2*exp(x)/(1+exp(x))-1`

**constr** FALSE

**nrow.ncol** FALSE

**augmented** FALSE

**aug.factor** 1

**aug.constr**

**n.div.by**

**n.required** FALSE

**set.default.values** FALSE

**pdf** ar

### Example I

```
n = 100L
p = 2L
pacf = runif(p)
phi = inla.ar.pacf2phi(pacf)
y = arima.sim(n, model = list(ar = phi)) +
    rnorm(n, sd=sd(y)/100.0)
idx = 1L:n

param.prec = c(1, 0.01)
param.psi.mean = rep(0, p)
param.psi.prec = 0.15 * diag(p)
param.psi = c(param.psi.mean, param.psi.prec)

r = inla(y ~ -1 + f(
        idx, model='ar',
        order = p,
        hyper = list(
                ## marginal precision
                prec = list(param = param.prec),
                ## the parameters for the joint normal prior for the
                ## transformed pacf's, goes here.
                pacf1 = list(param = param.psi))),
        family = "gaussian",
        data = data.frame(y, idx))

## we will now estimate the posterior marginals of the phi-parameters using
## 'inla.hyperpar.sampler', which creates samples from the approximated joint distribution for
## the hyperparameters.
nsamples = 100000
pacfs = inla.hyperpar.sampler(nsamples, r)[, 3L:(3L+(p-1L))]
phis = apply(pacfs, 1L, inla.ar.pacf2phi)
for(i in 1:p) {
    inla.dev.new()
    plot(density(phis[i, ]), main = paste("phi", i, sep=""))
    abline(v = phi[i])
}
```

### Example II

I this example we demonstrate how the parameters in the PC-prior for AR($p$) can be set using the idea in S. H. Søerbye and H. Rue (2016)[2].

```
library(gsl)

inla.pc.ar.lambda = function(a = 0.5, gamma = 0.8, p = 4)
{
    inla.pc.ar.solve.lambda = function(pred.err.factors, nseq = 1000L) {
        ## pred.err.factor = E(1-rho^2)
        pred.err = function(lambda) {
            return(0.5 * lambda * sqrt(pi) * exp(lambda^2/4 + log_erfc(lambda/2)))
        }

        ## find lower and upper limit of lambda for the pred.err.factors given
        lambda.min = lambda.max = 1
        val.max = max(pred.err.factors)
        val.min = min(pred.err.factors)
        while(pred.err(lambda.min) > val.min) {
            lambda.min = lambda.min / 2.0
        }
        while(pred.err(lambda.max) < val.max) {
            lambda.max = lambda.max * 2.0
        }

        lambda = lambda.min * exp(seq(0, log(lambda.max/lambda.min), length = nseq))
        fun = splinefun(pred.err(lambda), lambda,  method = "monoH.FC")
        lambdas = fun(pred.err.factors)

        return (lambdas)
    }

    pred.err.factors = 1.0 - (1.0-a)*gamma^(0:(p-1))
    lambda = inla.pc.ar.solve.lambda(pred.err.factors)

    return (lambda)
}

inla.pc.ar.test1 = function(p=1, p.est = p+1, n = 100)
{
    pacf = c(runif(p), rep(0, p.est-p))
    phi = inla.ar.pacf2phi(pacf)
    sd.x = sqrt(1/prod(1-pacf^2))
    x = arima.sim(n = n, model = list(ar = phi))/sd.x
    lambda = c(inla.pc.ar.lambda(p = p.est, gamma = 0.5), rep(1, 10))
    initial = c(inla.models()$latent$ar$hyper$theta2$to.theta(pacf), rep(0, 10))
    r = (inla(
        x ~ -1 +
```

[2]Penalised complexity priors for stationary autoregressive processes. arXiv preprint arXiv:1608.08941, UiT The Artic University of Norway, 2016

```
            f(time,  model = "ar",  order = p.est,
              hyper = list(
                  prec = list(param = c(3, 0.01), initial = 0),
                  pacf1 = list(param = c(lambda[1], 0), initial = initial[1]),
                  pacf2 = list(param = c(lambda[2], 0), initial = initial[2]),
                  pacf3 = list(param = c(lambda[3], 0), initial = initial[3]),
                  pacf4 = list(param = c(lambda[4], 0), initial = initial[4]),
                  pacf5 = list(param = c(lambda[5], 0), initial = initial[5]),
                  pacf6 = list(param = c(lambda[6], 0), initial = initial[6]),
                  pacf7 = list(param = c(lambda[7], 0), initial = initial[7]),
                  pacf8 = list(param = c(lambda[8], 0), initial = initial[8]),
                  pacf9 = list(param = c(lambda[9], 0), initial = initial[9]),
                  pacf10 = list(param = c(lambda[10], 0), initial = initial[10]))),
        data = data.frame(x=x, time = 1:length(x)),
        control.family = list(hyper = list(prec = list(initial = 12,
                                                       fixed=TRUE)))))
    result = cbind(est = r$summary.hyperpar$mean[-1], true = pacf)
    inside = c()
    for(i in 1:p.est) {
        int = inla.hpdmarginal(0.95, r$marginals.hyperpar[[i+1]])
        inside[i] = (int[1] < pacf[i] && pacf[i] < int[2])
    }
    result = cbind(result, coverage = inside)
    print(round(result, digits=3))

    result = (cbind(acf.est = inla.ar.pacf2acf(r$summary.hyperpar$mean[-1], lag.max = 10),
                    acf.emp = c(acf(x, lag.max=10, plot=FALSE)$acf),
                    acf.true = inla.ar.pacf2acf(pacf, lag.max=10)))
    print(round(result, digits=3))

    return (invisible())
}
```

### Notes

- The functions `inla.ar.pacf2phi` and `inla.ar.phi2pacf` converts from the the $\phi$-parameters to the $\psi$-parameters, using the Durbin-Levinson recursions. These can also be used to compute, the marginal posteriors of the $\phi$-parameters from an approximation of the joint of the $\phi$-parameters; see the example for a simulation based approach.
- Currently, the order $p$ is limited to 10. If this creates a problem, let us know.
- If some of the $\psi_k$-parameters are fixed, and $k < p$, then the marginal (log-)likelihood is wrong; The joint normal prior for all the $p$ $\psi$-parameters is used and not the conditional normal prior condition on the fixed $\psi_k$-parameters. If this creates a problem, let us know.
- The prior spesification for the multivariate normal is a bit awkward. Hopefully, we will come up with a better way to do this in the future.