Gibbs sampling and JAGS

Alberto Garfagnini

Università di Padova

AA 2020/2021 - Stat Lect. 12

Computational Bayesian Statistics

- the posterior distribution itself is the essence of bayesian inference

$$P(\theta \,|\, y) = \frac{f(y \,|\, \theta)\; g(\theta)}{\int f(y \,|\, \theta)\; g(\theta) d\,\theta}$$

- but most of the time it is not known analytically, and it must be computed numerically with Monte Carlo methods
- Markov Chain Monte Carlo (MCMC) methods are commonly used for sampling from a posterior distribution: we let the Markov chain *run* long enough ➜ a random draw from the chain can be considered a random draw from the posterior
- it's a radically different approach: instead of computing numerically the posterior distribution, we draw a sample from the posterior distribution.
- two main MCMC methods are commonly used:
  - the Metropolis-Hastings algorithm
  - the Gibbs sampling algorithm

Metropolis-Hastings : 1-dim example

- it samples from a target density by choosing values from a candidate density
- the acceptance of the new value (*proposal*) depends only on the previously accepted value (*current value*)
- using a symmetric transition probability, we generate a Markov Chain
- the acceptance probability, also called Metropolis ratio, is

$$\rho = \mathtt{MIN}\left(1,\ \frac{f(s)}{f(\theta_t)}\,\frac{Q(\theta_t \,|\, s)}{Q(s \,|\, \theta_t)}\right)$$

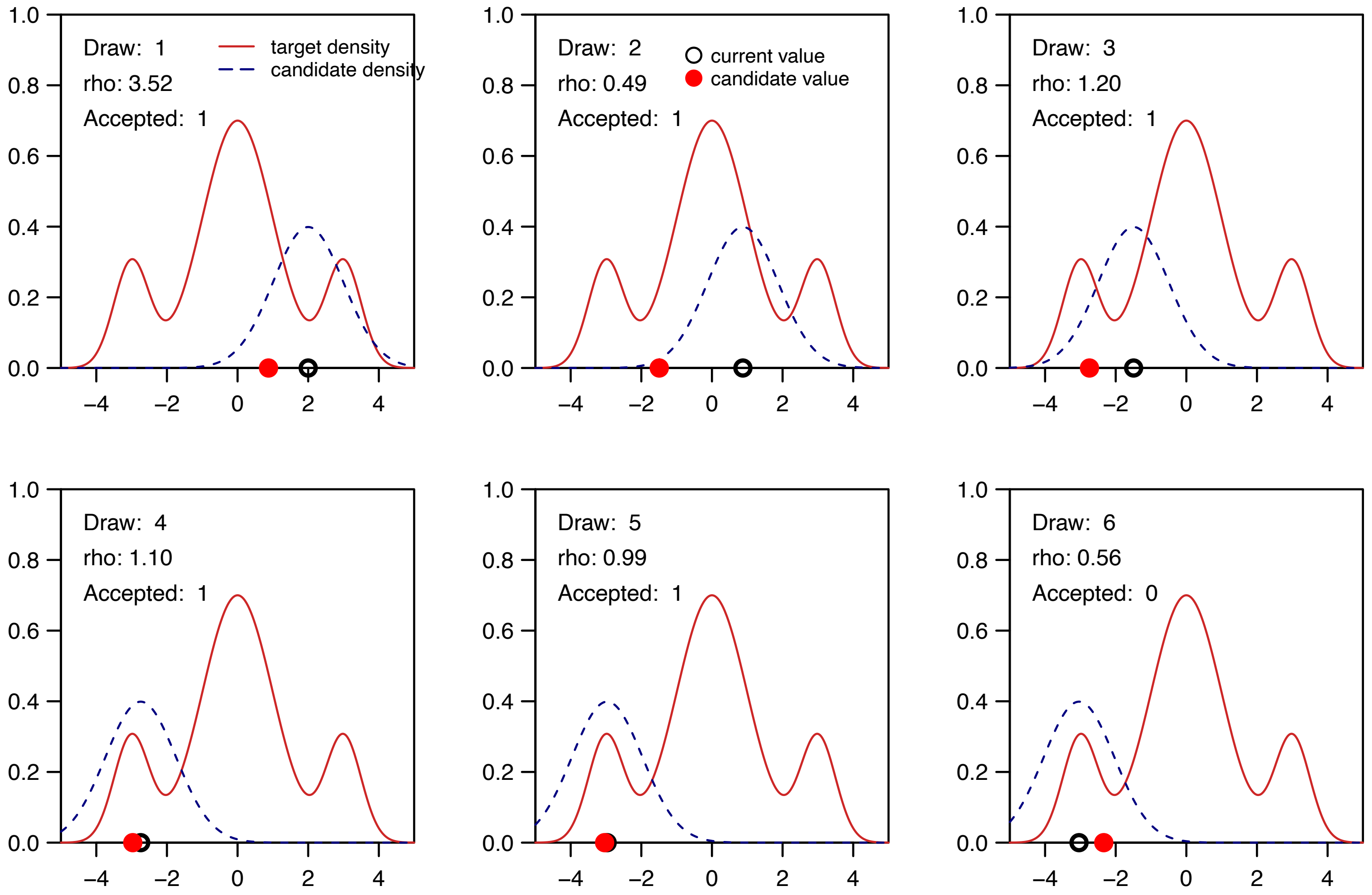
Problem

- let's assume ve have a target density that is the sum of three Normal distributions

$$f(x) = a_1\ \mathtt{Norm}(0,\ 1) + a_2\ \mathtt{Norm}(3,\ 0.5^2) + a_3\ \mathtt{Norm}(-3,\ 0.5^2)$$

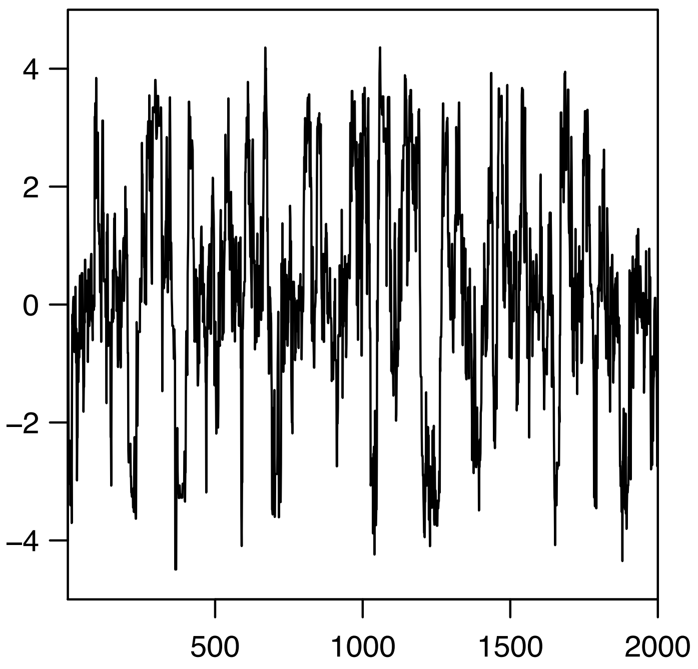
- with $a_1 = 0.7, a_2 = 0.15$ and $a_3 = 0.15$

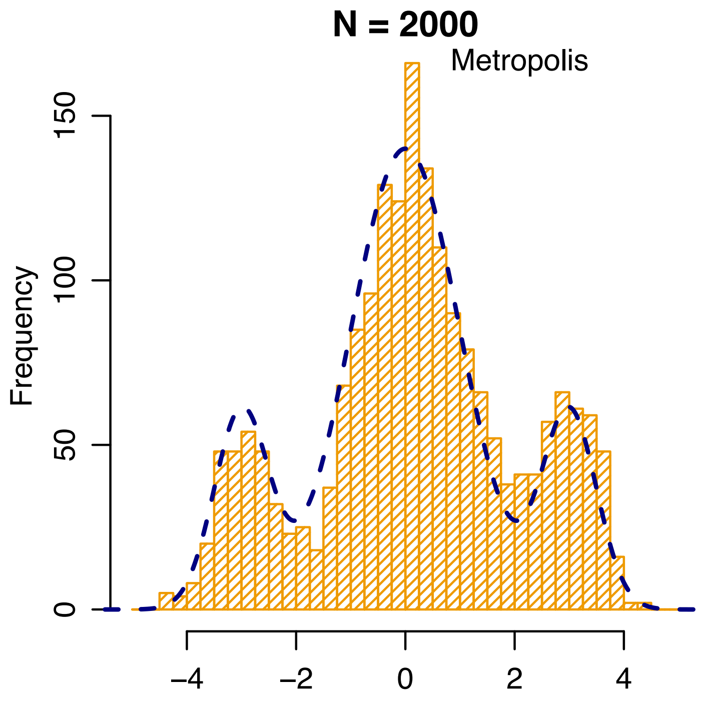
Metropolis-Hastings : 1-dim example



Metropolis-Hastings : 1-dim example

- the sample is moving through the space quite satisfactory
- extreme values are selected, from time to time, but the chain tends to jump back to the central region (with higher probability) very quickly





Metropolis-Hastings : indep. cand. chain

- a variant to the Metropolis-hastings algorithm uses an independent candidate density
- Hastings (1970) introduced Markov Chains with candidate densities that did not depend on the current value in the chain

$$Q(s \mid \theta) = Q_2(s)$$

- Q_2 is some function that dominates the target density in the tails
- therefore the acceptance probability, the Metropolis ratio, simplifies to

$$\rho = \texttt{MIN}\left(1,\ \frac{f(s)}{f(\theta_t)}\frac{Q(\theta_t \mid s)}{Q(s \mid \theta_t)}\right) = \texttt{MIN}\left(1,\ \frac{f(s)}{f(\theta_t)}\frac{Q_2(\theta_t)}{Q_2(s)}\right)$$

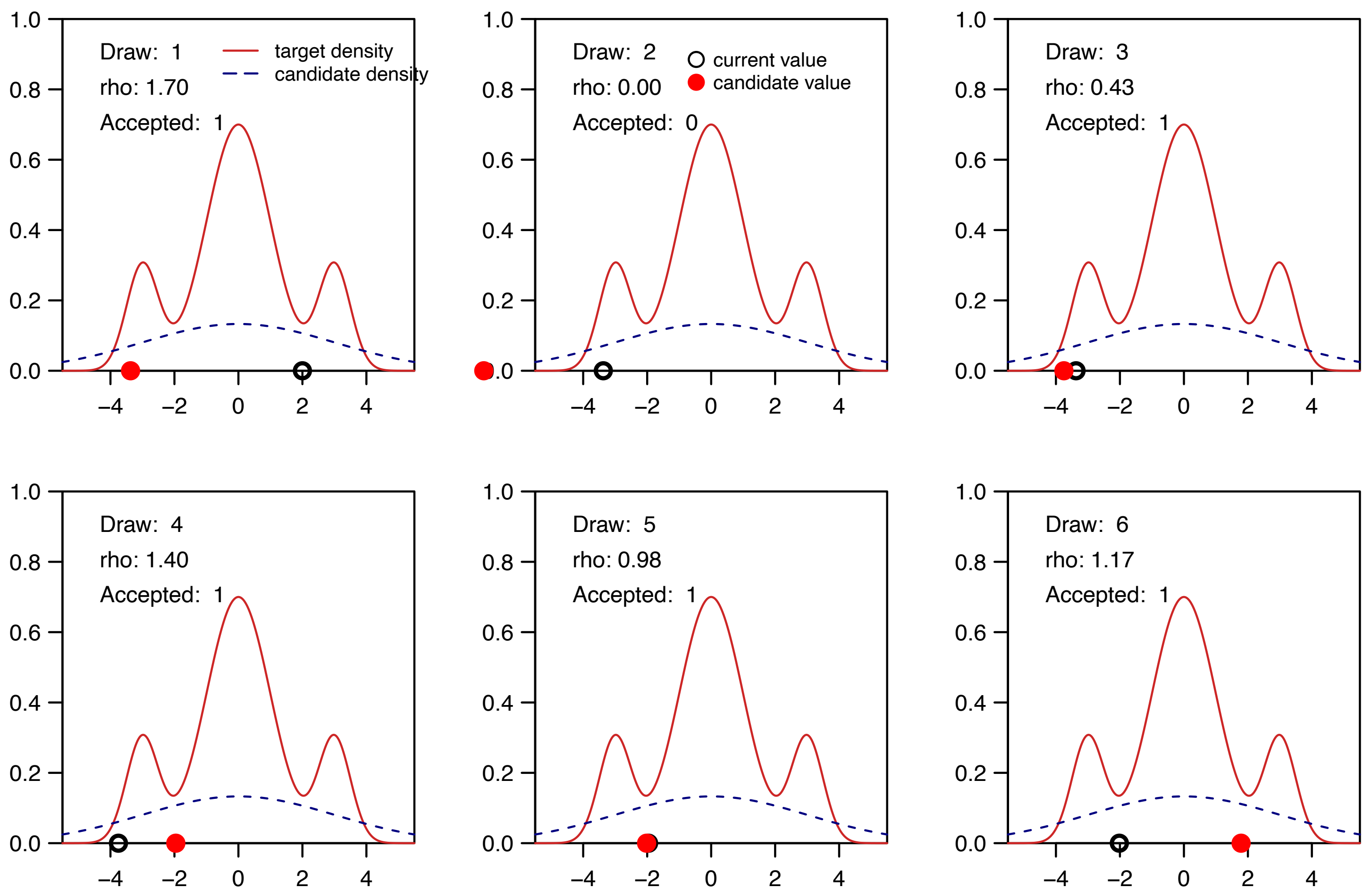
Problem

- let's study the same problem

$$f(x) = a_1\,\texttt{Norm}(0,\ 1) + a_2\,\texttt{Norm}(3,\ 0.5^2) + a_3\,\texttt{Norm}(-3,\ 0.5^2)$$

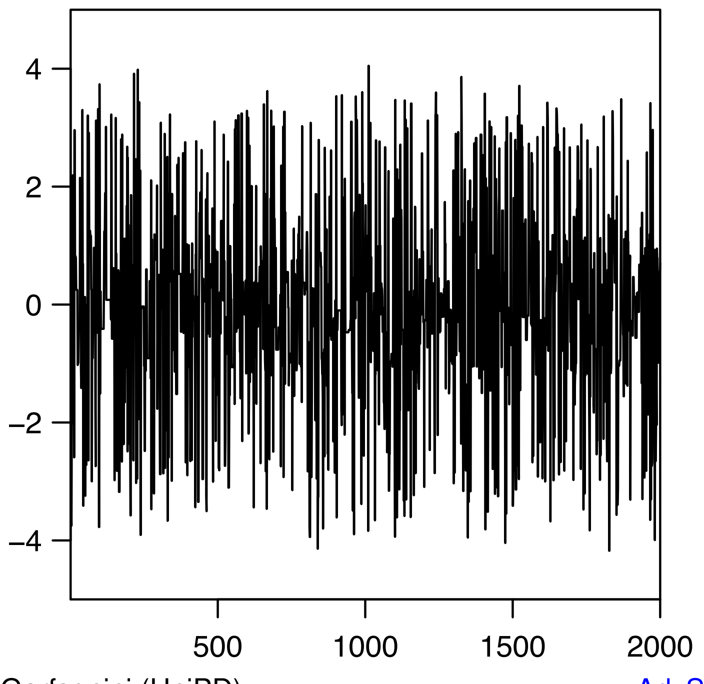
- with $a_1 = 0.7, a_2 = 0.15$ and $a_3 = 0.15$
- assuming that the candidate density is a $\texttt{Norm}(0, 3^2)$ distribution function

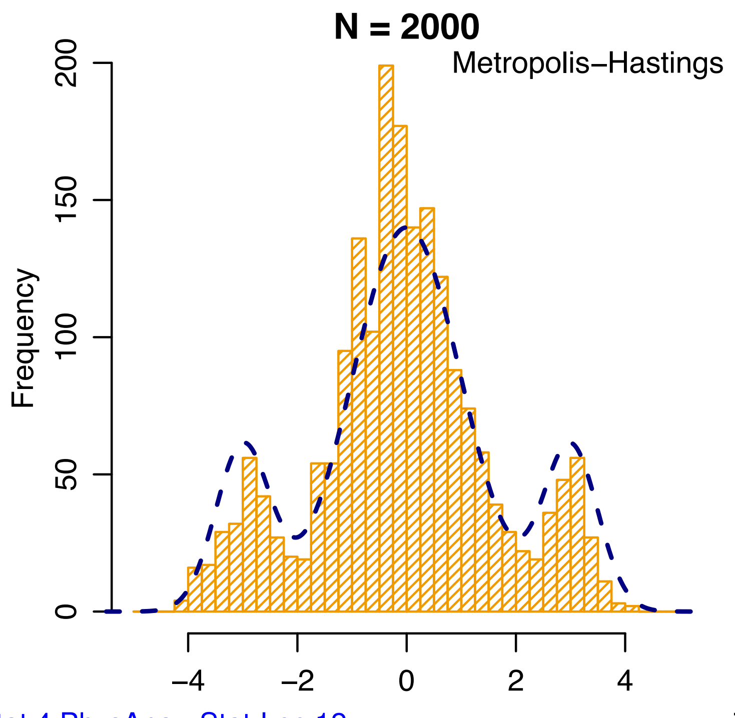
Metropolis-Hastings : indep. cand. chain



Metropolis-Hastings : indep. cand. chain

- the sample is moving through the space quite satisfactory
- the independent candidate density allows for larger jumps, but it may accept fewer proposals than the random-walk chain
- nevertheless the acceptance is larger and the chain will potentially explore the parameter space faster





Gibbs sampling

- it is one of the most widely used algorithms for simulating Markov chains
- it is a special case of the Metropolis-Hastings algorithm and it is most relevant with multi-parameters problems
- in general, Metropolis-hasting can be improved by only updating a block of parameters at each iteration ➜ blockwise Metropolis-Hastings algorithm
- the Gibbs sampling algorithm is a special case of the blockwise Metropolis-Hastings
- it generates a multi-dimensional Markov chain by splitting the vector of random variables θ into subvectors and sampling each subvector in turn, conditional on the most recent values of all other elements of θ
- the beauty of Gibbs sampling is that simulation from a complex, high-dimensional joint posterior distribution is reduced to a sequence of algorithms for sampling from one or low-dimensional distributions
- Gibbs sampling is most suited for hierarchical models, where the dependencies between model parameters is well-defined

Gibbs sampling algorithm

(1) choose arbitrary starting values $\theta_1^{(0)}, \theta_2^{(0)}, \dots, \theta_k^{(0)}$
(subscript = component, superscript = iteration step)

(2) sample new values for each element with the following steps:

- sample $\theta_1^{(1)}$ from the full-conditional distribution,

$$\theta_1^{(1)} \sim P(\theta_1 \,|\, \theta_2^{(0)}, \theta_3^{(0)}, \dots \theta_k^{(0)}, y)$$

where y indicates the data

- sample a new $\theta_2^{(1)}$, for the second component, from its full conditional distribution

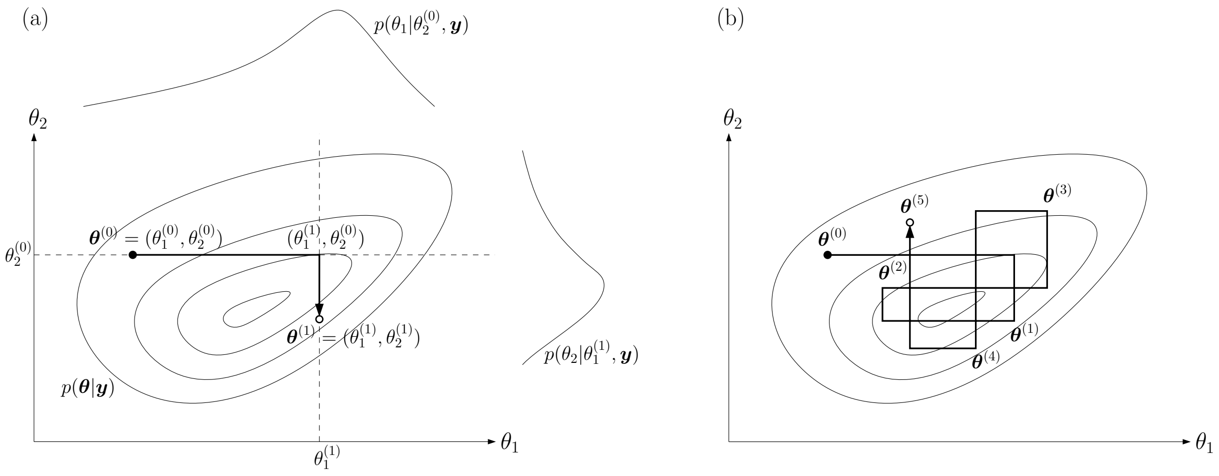
$$\theta_2^{(1)} \sim P(\theta_2 \,|\, \theta_1^{(1)}, \theta_3^{(0)}, \dots \theta_k^{(0)}, y)$$

- complete the step for all the other components, obtaining a sequence of dependent realization of $\theta_1^{(1)}, \theta_2^{(1)}, \dots, \theta_k^{(1)}$

(3) repeat `step 2` many times conditioning on the most recent values of other parameters

Gibbs sampling algorithm : 2-dim example

- picture (b) shows the first five iterations of the Gibbs sampler
- the sampler always moves parallel to the axes
- the starting point, $\theta^{(0)} = (\theta_1^{(0)}, \theta_2^{(0)})$ is shown
- $p(\theta_1 | \theta_2^{(0)}, y)$, shown on top, is the univariate density and is obtained by taking a horizontal "slice" through the 2 joint posterior distribution at the value $\theta_2 = \theta_2^{(0)}$ (horizontal dashed line)
- a new value for $\theta_1^{(0)}$ is generated from this full conditional, and then a "slice" parallel to the θ_2 axis is taken through the joint posterior (vertical dashed line)


(a)
$p(\theta_1|\theta_2^{(0)}, \boldsymbol{y})$
θ_2
$\theta_2^{(0)}$
$\boldsymbol{\theta}^{(0)} = (\theta_1^{(0)}, \theta_2^{(0)})$
$(\theta_1^{(1)}, \theta_2^{(0)})$
$\boldsymbol{\theta}^{(1)} = (\theta_1^{(1)}, \theta_2^{(1)})$
$p(\theta_2|\theta_1^{(1)}, \boldsymbol{y})$
$p(\boldsymbol{\theta}|\boldsymbol{y})$
θ_1
$\theta_1^{(1)}$
(b)
θ_2
$\boldsymbol{\theta}^{(0)}$
$\boldsymbol{\theta}^{(5)}$
$\boldsymbol{\theta}^{(3)}$
$\boldsymbol{\theta}^{(2)}$
$\boldsymbol{\theta}^{(1)}$
$\boldsymbol{\theta}^{(4)}$
θ_1

JAGS: Just Another Gibbs Sampler

- JAGS is a program mainly written by M. Plummer with the aim of providing a BUGS (Bayesian Inference Using Gibbs Sampling) engine for UNIX
  - more infos are available at `http://sourceforge.net/projects/mcmc-jags/`
  - the latest version is 4.3.0 (July 18, 2017)

- `rjags` is another `R` package that allows to run JAGS from within `R`
  `https://cran.r-project.org/web/packages/rjags/`
  `https://cran.r-project.org/web/packages/rjags/rjags.pdf`
  - available for Linux-64 (`v4.6`) and osx-64 (`v4.6`)

```
conda install -c conda-forge r-rjags
```

- `R2jags` is an `R` package that allow to fit JAGS models from within `R`
  `https://cran.r-project.org/web/packages/R2jags/`
  `https://cran.r-project.org/web/packages/R2jags/R2jags.pdf`
  - available only for Linux-64 (`v0.5_7`)

```
 conda install -c glaxosmithkline r-r2jags
```

`rjags`

- an analysis with `rjags` proceeds through the following steps:

(1) define the model using the BUGS language in a separate file

(2) read in the model file using the `jags.model` function. This creates an object of class `jags`

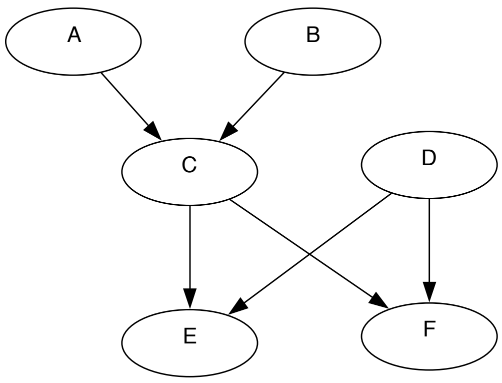
(3) update the model using the update method for `jags` objects.
This constitutes the *burn-in* part

(4) extract samples from the model object using the `coda.samples` function. This creates an object of class `mcmc.list` which can be used to summarize the posterior distribution. The `coda` package also provides convergence diagnostics to check that the output is valid for analysis

The BUGS language

- BUGS (Bayesian inference Using Gibbs Sampling) is also a language that allows to specify Bayesian models for Bayesian computation
- it is based on graphical representation which is used to express the joint relationship between all known and unknown quantities in a model through a series of simple local relationships

- let's consider the graph in the figure:
- A, B and D have no parents and are therefore marginally independent
- A and B are parents of C which, in turn, is a parent (with D) of E and F
- if we observe E, this will induce a dependency between C and D and between A and B, since two nodes without common parents are only independent given no descendants have been observed
- from the graph we can see that the joint distribution of the set of quantities may be written



$$P(A,B,C,D,E,F) = P(A)P(B)P(C|A,B)P(D)P(E|C,D)P(F|C,D)$$

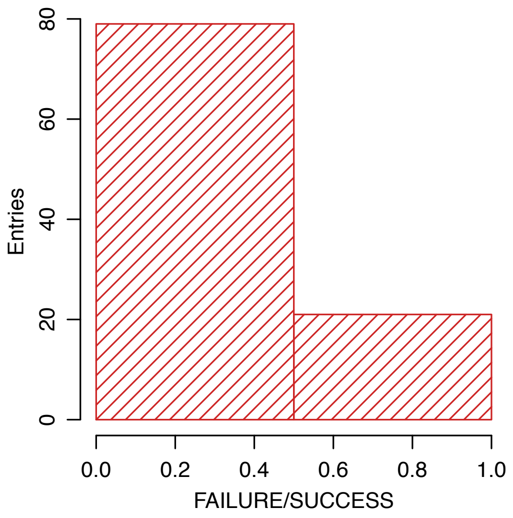
Ex 1: Bernoulli process

The Problem

- given a set of observation, coming from a Bernoulli process, we want to infere the probability *p* of the process from the sequence of success/failure, and predict the number of successes in the future

- the observed sequence is the following:

```
X <- c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1)
```



- we describe the model with BUGS and let `jags` solve our inference problem

Ex 1: BUGS model and parameters

- file: `s13_inf_p_pred.bug`

```
model {
   # data likelihood
   for (i in 1:length(X)) {
     X[i] ˜ dbern(p);
   }
   # a uniform prior for p
   p ˜ dbeta(1, 1);

   # Predicted data, given p
   y ˜ dbin(p, n_next);
}
```

- a list with the data for the model :

```
data <- NULL
data$X <- data_obs      # Set of observations
data$n <- length(X)     # those to be considered

data$n_next <- 10       # Predictions
```

- the model is created passing the BUGS data file and a list with all the data and model parameters

```
jm <- jags.model(model, data)
```

Ex 1: running jags

```
# Update the Markov chain (Burn-in)
update(jm, 1000)
chain <- coda.samples(jm, c("p", "y"), n.iter=10000)
print(summary(chain))
```

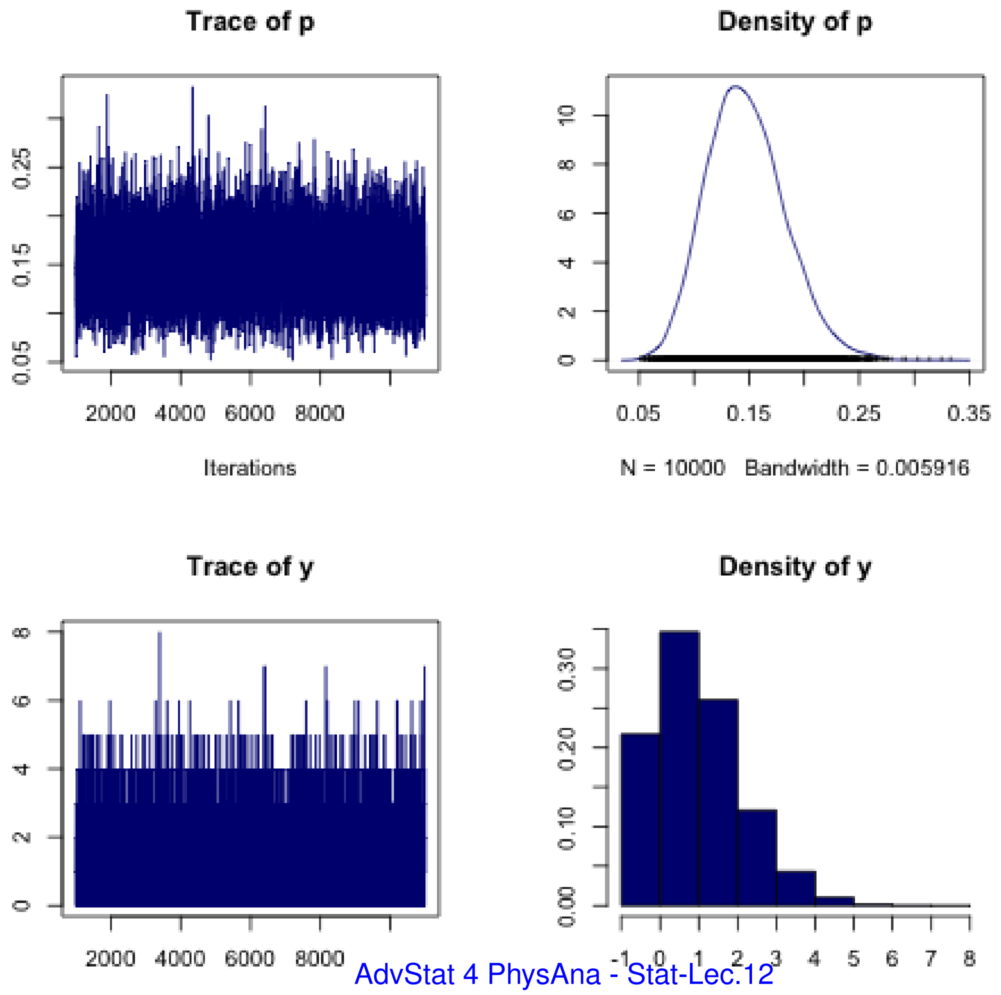
- Output from R:

```
Compiling model graph
   Resolving undeclared variables /  Allocating nodes
Graph information:
   Observed stochastic nodes: 100 / Unobserved stochastic nodes: 2
   Total graph size: 105

Initializing model
  |**************************************************| 100%
  |**************************************************| 100%

Iterations = 1001:11000
Thinning interval = 1 / Number of chains = 1 | SampSize/chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
    Mean       SD  Naive SE Time-series SE
p 0.1474 0.03495 0.0003495      0.0003547
y 1.4872 1.17489 0.0117489      0.0117489

2. Quantiles for each variable:
     2.5%     25%     50%     75%   97.5%
p 0.08681 0.1225 0.1448 0.1695 0.2224
y 0.00000 1.0000 1.0000 2.0000 4.0000
```

Ex 1: producting control plots

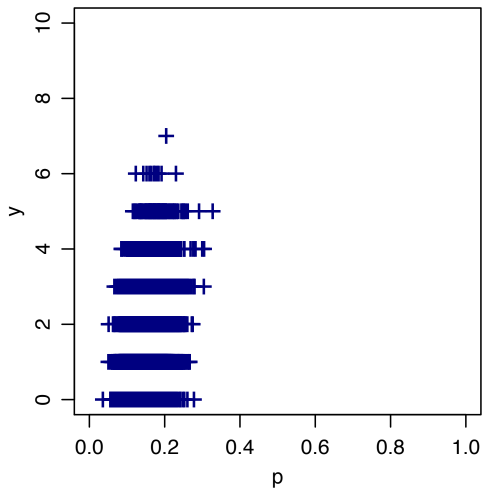
```
plot(chain, col="navy")

# Let's format our chain
chain.df <- as.data.frame( as.mcmc(chain) )
cat(sprintf("\n␣Correlation␣matrix:␣\n"))
print(cor(chain.df))

#
# p inference result
#
hist(chain.df$p, nc=50, prob=TRUE, col='darkolivegreen2',
     xlab='p', ylab='f(p)', main='Inference␣on␣p')

#
# next data prediction probability
#
ty <- table(chain.df$y)
barplot(ty/sum(ty), col='firebrick2', xlab='y', ylab='f(y)',
        ylim=c(0,0.40),
        main=sprintf('Number␣of␣successes␣in␣%d␣future␣trials', data$n_next))

#
# Correlation between p and predicted variable
#
plot(chain.df$p, chain.df$y, xlab='p', ylab='y', main="",
     pch='+', col='navy', cex=1.5,
     xlim=c(0,1), ylim=c(0,10))
```

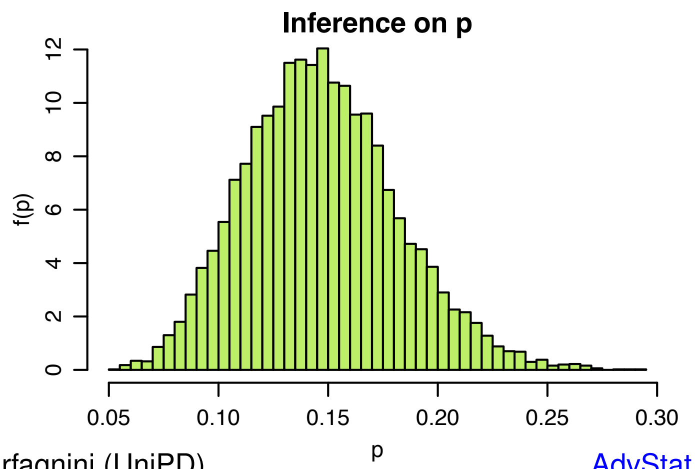
Ex 1: jags chains

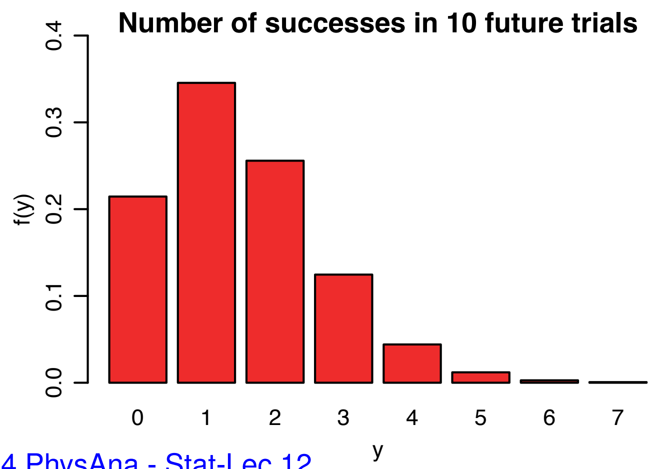


Ex 1: jags results

```
    Mean       SD  Naive SE Time-series SE
p 0.1474 0.03495 0.0003495      0.0003547
y 1.4872 1.17489 0.0117489      0.0117489

     2.5%     25%     50%     75%   97.5%
p 0.08681 0.1225 0.1448 0.1695 0.2224
y 0.00000 1.0000 1.0000 2.0000 4.0000

 Correlation matrix:
           p         y
p 1.0000000 0.3031662
y 0.3031662 1.0000000
```







Ex 2: Poisson inference

The Problem

- given the number of counts from a ionizing radiation detector, we want to infer the parameter λ of the underlying Poisson process
- the BUGS model (file: s13_inf_lambda_pred.bug) is the following:

```
model {
   # data likelihood
   X ~ dpois(lambda);

   # a uniform prior for lambda
   lambda ~ dexp(0.00001)

   # Predicted data, given lambda
   Y ~ dpois(lambda);
}
```

- and our data:

```
data <- NULL
data$X <- 100  # number of counts
```

- we create the jags model:

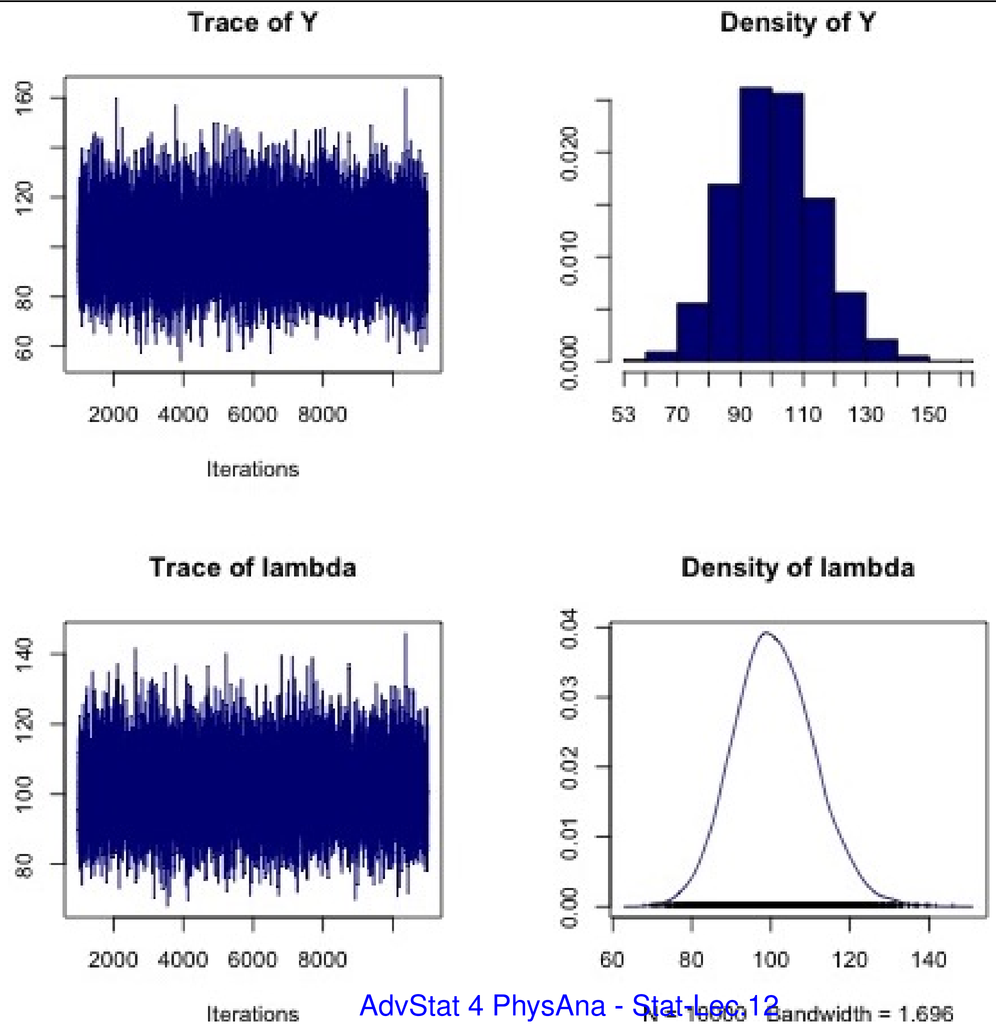
```
library(rjags)
model <- "s13_inf_lambda_pred.bug"
jm <- jags.model(model, data)
```

Ex 2: Poisson inference

- the rest of the code is:

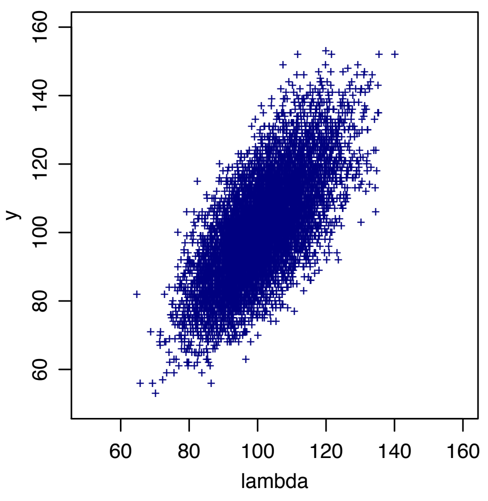
```
# Update the Markov chain (Burn-in)
update(jm, 1000)

chain <- coda.samples(jm, c("lambda", "Y"), n.iter=10000)

plot(chain, col="navy")

# Let's format our chain
chain.df <- as.data.frame( as.mcmc(chain) )

#
# Probability plots
par(mfrow=c(3,2), mgp=c(2.0,0.8,0), mar=c(3.5,3.5,1,1), oma=0.1*c(1,1,1,1))
hist(chain.df$lambda, nc=100, prob=TRUE, col='darkolivegreen2',
     xlim=c(40, 170),
     xlab='lambda', ylab='f(lambda)', main='Inference␣on␣lambda')

ty <- table(chain.df$Y)
barplot(ty/sum(ty), col='firebrick2', xlab='Y', ylab='f(Y)',
        # ylim=c(0,0.40),
        main=sprintf('Predicted␣counts'))

#
# And present/ future prediction correlations
plot(chain.df$lambda, chain.df$Y, xlab='lambda', ylab='y', main="",
     pch='+', col='navy', cex=0.75, asp=1,
     xlim=c(50,160), ylim=c(50,160))
```

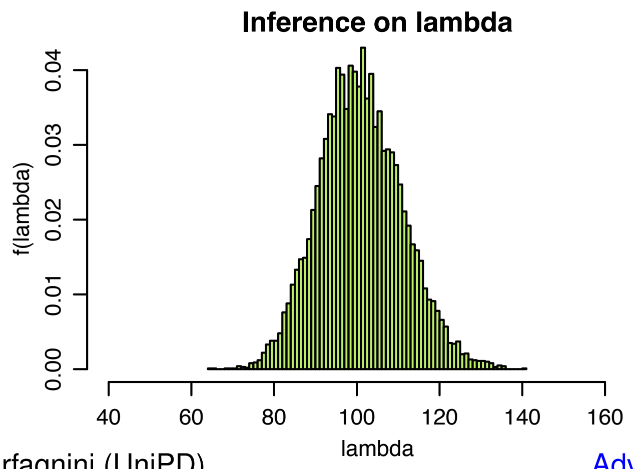
Ex 2: `jags` chains

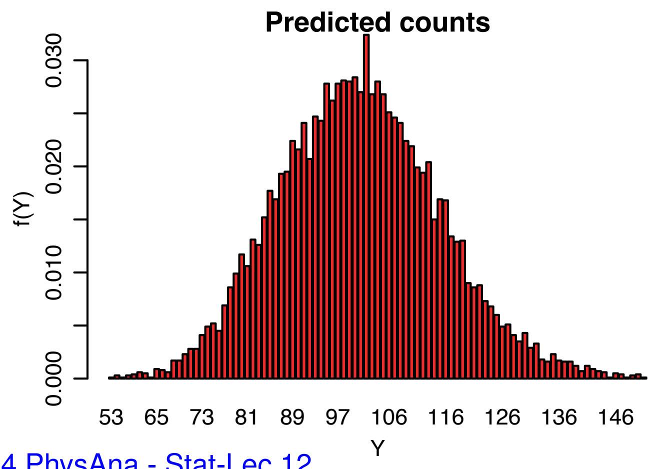


Ex 2: `jags` Poisson results

```
1. Empirical mean and standard deviation for
   each variable,
   plus standard error of the mean:

         Mean    SD  Naive SE  Time-series SE
Y       101.1 14.26    0.1426          0.1426
lambda  100.9 10.10    0.1010          0.1010

2. Quantiles for each variable:

         2.5%   25%   50%   75% 97.5%
Y       75.00 91.00 101.0 110.0 131.0
lambda  82.24 93.99 100.5 107.6 121.7
```





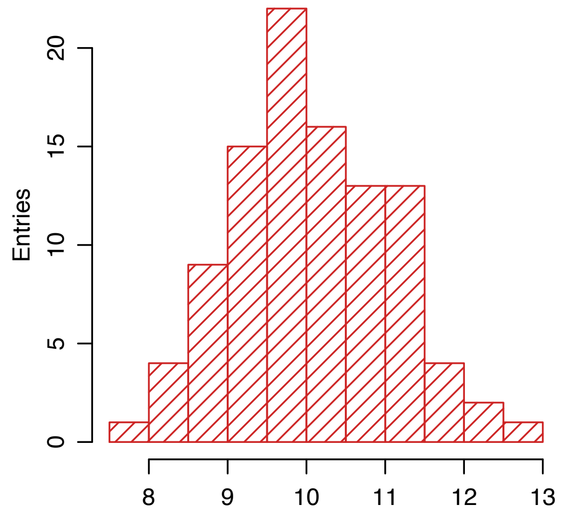


Ex 3: Normal inference

Problem

- given a set of 100 measurements, we want to infer the mean and sigma, assuming they are coming from a gaussian distribution with unknown mean and sigma
- the BUGS model (file: s13_norm_pred.bug) is the following:

```
#
# Gaussian model with unknown mean and sigma
#
model {
    for (i in 1:length(X)) {
        X[i] ~ dnorm(mu, tau);
    }
    mu ~ dnorm(0.0, 1.0E-6);

    tau ~ dgamma(1.0, 1.0E-4);
    sigma <- 1.0/sqrt(tau);

    # future observation
    Y ~  dnorm(mu, tau);
}
```



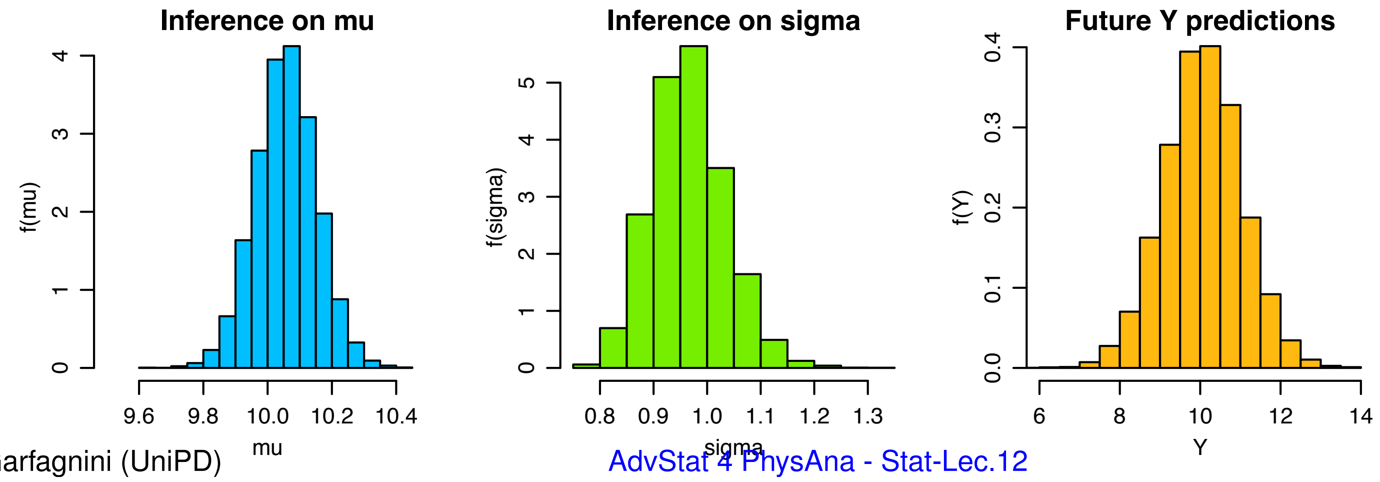
Ex 3: Normal inference

```
library(rjags)

set.seed(20190522)

#
# Generate the observed data
data_size <- 100
data_mu <- 10
data_sigma <- 1
data_obs <- rnorm(data_size, data_mu, data_sigma)

# - Specify the Generative Model with BUGS
model <- "s13_norm_pred.bug"

# Our data for the model
data <- NULL
data$X <- data_obs    # Set of observations

# Create the model and pass the parameters
jm <- jags.model(model, data)

# Update the Markov chain (Burn-in)
update(jm, 1000)

chain <- coda.samples(jm, c("mu", "sigma", "Y"), n.iter=10000)
print(summary(chain))
```

Ex 3: `jags` Normal results

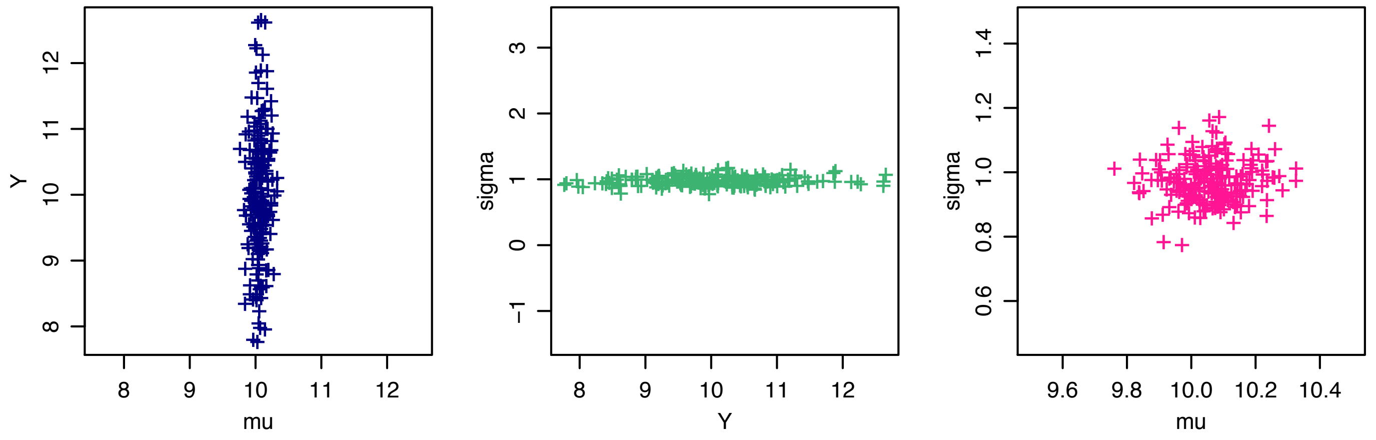
```
          Mean        SD  Naive SE Time-series SE
Y      10.0655 0.97382 0.0097382      0.0097382
mu     10.0577 0.09610 0.0009610      0.0009610
sigma   0.9656 0.06926 0.0006926      0.0006926

         2.5%     25%     50%    75%   97.5%
Y      8.1399 9.4278 10.067 10.711 11.983
mu     9.8668 9.9945 10.058 10.122 10.247
sigma 0.8398 0.9176  0.962  1.009  1.111
```

Note the different x-axis limits for μ and future Y predictions



Ex 3: `jags` Normal variables correlations

```
 Correlation matrix:
                Y          mu       sigma
Y     1.000000000 0.101044364 0.008388187
mu    0.101044364 1.000000000 0.002831711
sigma 0.008388187 0.002831711 1.000000000
```



Ex 4: Hook’s law inference

The Problem

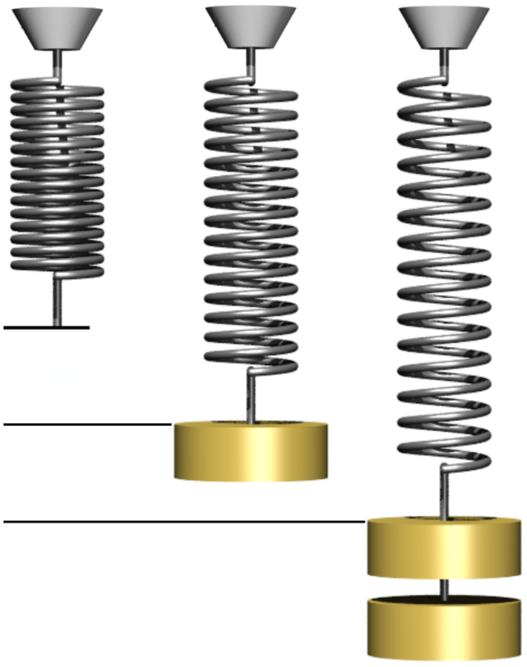
- a spring with elastic constant k and mass m_{spring} is held vertically, at rest, under the influence of the Earth’s gravitational field
- the lower end of the spring is loaded with equal mass discs and both spring elongation and oscillation periods are measured
- we want to infer, from the data, the spring elastic constant, k and, eventualy, the Earth gravity constant, g
- calling $l_\circ$, the unloaded spring length, we get at equilibrum

$$l = l_\circ + \frac{g}{k}(m_{spring} + n \cdot m_{disc})$$

where m_{disc} is a disc mass and n the number of discs connected to the spring

- if one end of the spring is perturbated from the equilibrium position, it oscillates with period

$$T = 2\pi\sqrt{M/k} \text{ where } M = m_{spring} + n \cdot m_{disc}$$



measuring the oscillation period. as a function of the applied mass, it is possible to measure k, and from l, infer g

Ex 4: the collected data

- the following data come from https://www.roma1.infn.it/~dagos/BMS/node22.html

		I series		II series		III series	
n	M (g)	l (mm)	$T \times 10$ (s)	l (mm)	$T \times 10$ (s)	l (mm)	$T \times 10$ (s)
0	63	0	-	0	-	0	-
1	142	0	-	0	-	0	-
2	221	0	-	0	-	0	-
3	300	14	5.01	16	5.09	16	5.19
4	379	32	5.57	33	5.66	33	5.68
5	458	49	6.24	51	6.27	51	6.34
6	536	66	6.78	68	6.82	69	6.94
7	615	85	7.28	86	7.33	87	7.28
8	694	103	7.79	103	7.81	103	7.86
9	773	119	8.13	121	8.31	121	8.24
10	852	137	8.63	139	8.77	139	8.70

- Notes: $l_\circ$, the unloaded spring rest length has been subtracted from data. Since the oscillation priod is below 1 s, the measurements have been taken for 10 periods

Ex 4: the BUGS model

```
model {
   # l Vs m
   for (i in 1:length(l)) {
      mu.l[i] <- c.l + m.l * (m_spring + (Nmin-1 + i) * m_disc);
      l[i] ~dnorm(mu.l[i], tau.l);
   }
   c.l ~ dnorm(0.0, 1.0E-4);
   m.l ~ dnorm(0.0, 1.0E-4);

   tau.l ~ dgamma(1.0E-3, 1.0E-6);
   sigma.l <- 1/sqrt(tau.l);

   # t vs sqrt(m)
   for (i in 1:length(t)) {
      mu.t[i] <- c.t + m.t * sqrt(m_spring + (Nmin-1 + i) * m_disc);
      t[i] ~dnorm(mu.t[i], tau.t);
   }
   c.t ~ dnorm(0.0, 1.0E-4);
   m.t ~ dnorm(0.0, 1.0E-4);

   tau.t ~ dgamma(1.0E-3, 1.0E-5);
   sigma.t <- 1/sqrt(tau.t);

   # k e g
   k <- 4*pi2 / (m.t*m.t)
   g <- m.l * k
}
```

Ex 4: data, init values, and model running

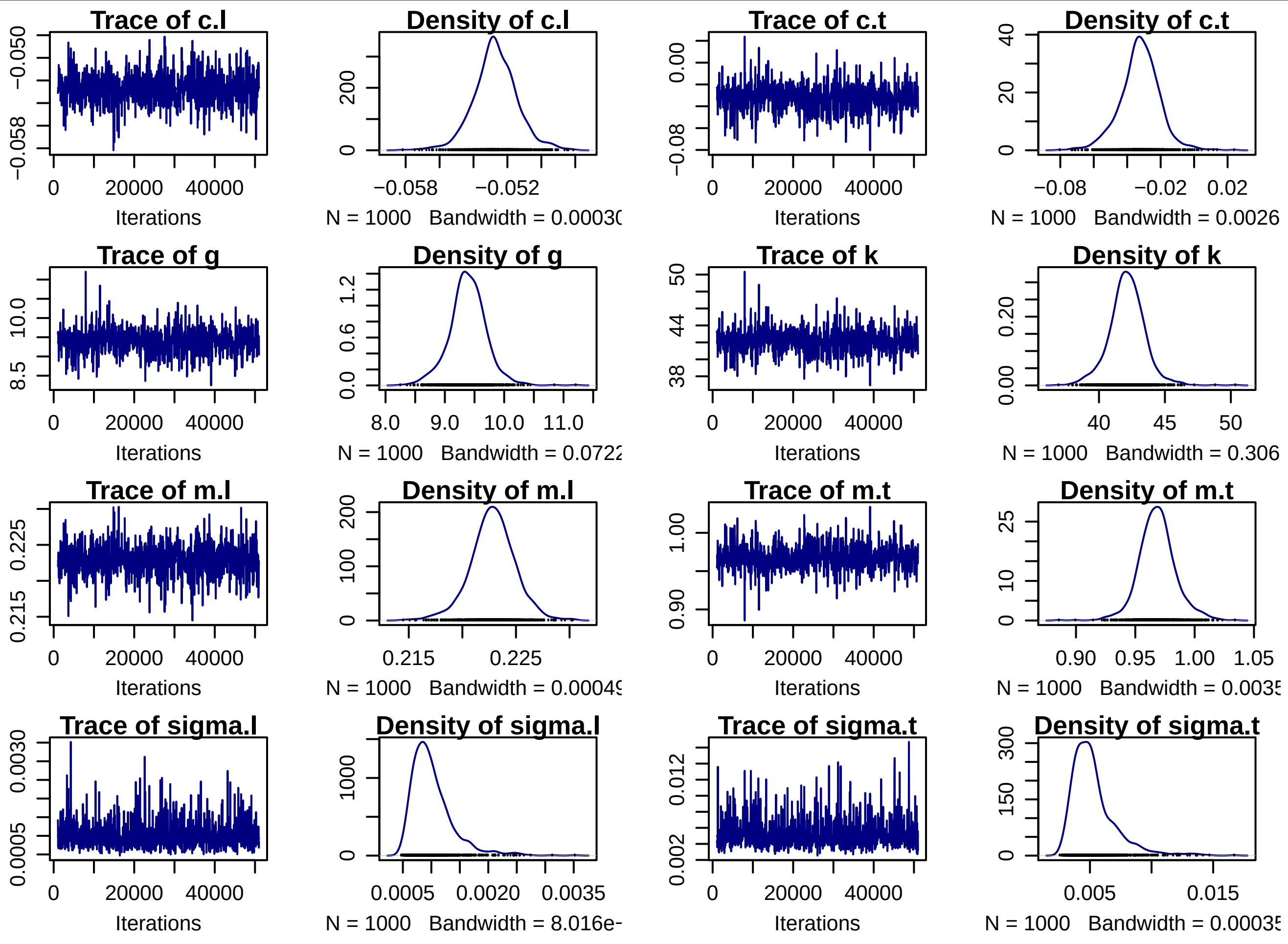
```
# Experimental data - Series I
data <- NULL
data$m_spring <- 0.063
data$m_disc   <- 0.0789
data$l        <- c(0.014, 0.032, 0.049, 0.066, 0.085, 0.103, 0.119, 0.137)
data$t        <- c(0.501, 0.557, 0.624, 0.678, 0.728, 0.779, 0.813, 0.863)
data$Nmin     <- 3
data$pi2      <- pi^2

# Generative model initial values
inits <- NULL
inits$c.l   <- 0
inits$m.l   <- 0
inits$tau.l <- 1000
inits$tau.t <- 1000
inits$m.t   <- 1

# Create the model and pass the parameters
jm <- jags.model("s13_spring.bug", data, inits)

# Update the Markov chain (Burn-in)
update(jm, 1000)

chain <- coda.samples(jm, c("c.l","m.l","sigma.l","c.t",
                            "m.t","sigma.t","k", "g"),
                      n.iter = 50000, thin = 50)
```

Ex 4: `jags` run results

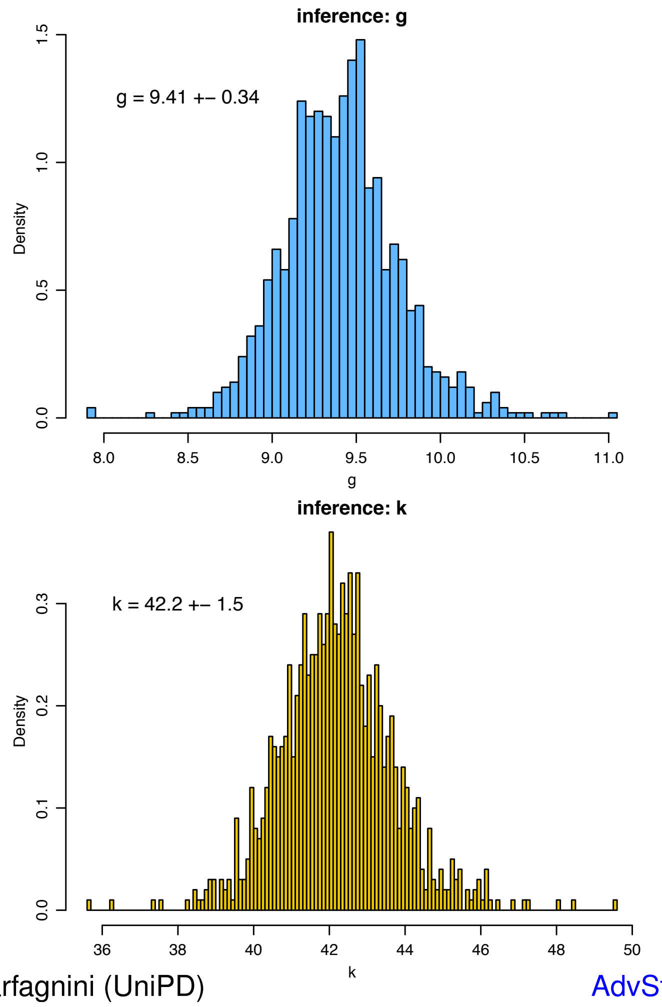
- the model produces the following output:

```
Iterations = 1050:51000 / Thinning interval = 50 / Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:
                 Mean        SD  Naive SE Time-series SE
c.l      -0.0527146 0.0012529 3.962e-05      3.962e-05
c.t      -0.0313581 0.0141587 4.477e-04      5.596e-04
g         9.4168338 0.3718726 1.176e-02      1.502e-02
k        42.2535756 1.6111324 5.095e-02      6.667e-02
m.l       0.2228622 0.0020382 6.445e-05      6.445e-05
m.t       0.9671275 0.0184045 5.820e-04      7.612e-04
sigma.l   0.0009943 0.0003414 1.080e-05      1.080e-05
sigma.t   0.0056743 0.0022657 7.165e-05      9.858e-05

2. Quantiles for each variable:
               2.5%        25%        50%        75%      97.5%
c.l      -0.0551276 -0.0534807 -0.0527385 -0.051996 -0.050116
c.t      -0.0622894 -0.0386805 -0.0312846 -0.023451 -0.003571
g         8.6816215  9.2058887  9.4212113  9.621637 10.127262
k        38.9954677 41.3512168 42.2117668 43.125521 45.426046
m.l       0.2188010  0.2216423  0.2229252  0.224129  0.226905
m.t       0.9322393  0.9567813  0.9670816  0.977093  1.006173
sigma.l   0.0005608  0.0007633  0.0009172  0.001125  0.001863
sigma.t   0.0030946  0.0042109  0.0050638  0.006621  0.011691
```

Ex 4: Markov chains plots



Ex 4: Inference, *g* and *k* results



```
sch <- summary(chain)
names(sch)
[1] "statistics" "quantiles"
"start"       "end"         "thin"
[6] "nchain"

> sch$statistics[,1:2]
                Mean
SD
c.l     -0.052742040 0.0013533757
c.t     -0.031768656 0.0131463627
g
9.407672249 0.3556819257
k       42.203673989 1.5186476007
m.l
0.222907961 0.0022597906
m.t
0.967640043 0.0173119299
sigma.l
0.001029015 0.0003875593
sigma.t
0.005411466 0.0018208406

sprintf("g␣=␣%.2f␣+-␣%.2f",
        sch$statistics["g","Mean"],
        sch$statistics["g","SD"])
[1] "g␣=␣9.41␣+-␣0.36"
```

References

Exercises

- http://www.roma1.infn.it/~dagos/prob+stat.html

Reference Books

- C.P. Robert and G.Casella, *Introducting Monte Carlo Methods with R*, Springer, 2010
- C.P. Robert and G.Casella, *Monte Carlo Statistical methods*, Springer, 1999
- D. Lunn, et. al., *The BUGS Book, a prctical introduction to Bayesian Analysis*, CRC Press, 2012

Additional Material

- JAGS user manual: https://sourceforge.net/projects/mcmc-jags/files/Manuals/
- rjags user manual https://cran.r-project.org/web/packages/rjags/rjags.pdf