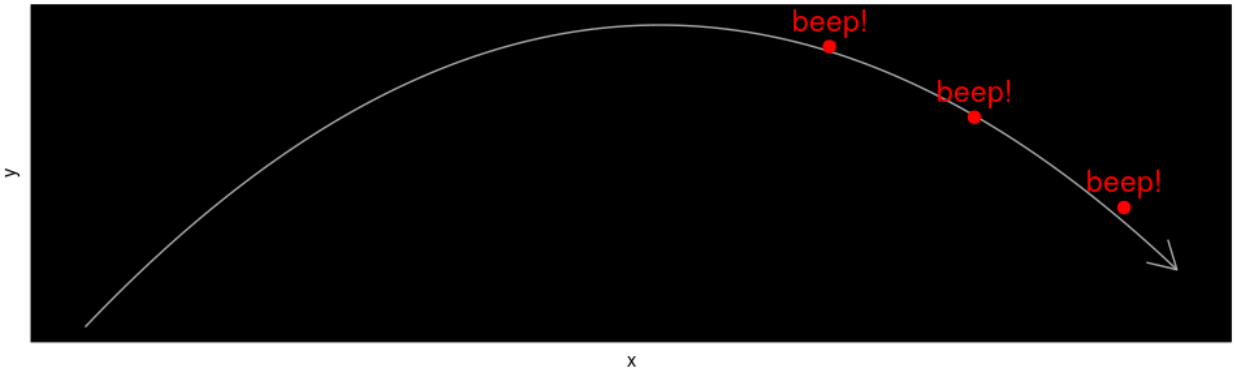
Bayesian Artillery

Alberto Dell'Era

A short summary is available at:
https://github.com/alberto-dellera/bayesian_artillery/blob/master/README.md
Github readers: github on-line resolution is very low, I'd suggest to download this pdf and open it

A battery is shooting at us, and all we have are some noisy radar readings about the projectile trajectory:



we want to find the battery position, also factoring in some prior knowledge provided by our Intelligence.

Summary

We will perform a standard Bayesian analysis (model building, MCMC by JAGS, chains exploration, etc), but from the practical perspective of the physical (mechanical) domain expert that wants to understand how the algorithm "reasons" about trajectories, projectile speed, etc etc.

In particular, we will explore how the Bayesian system incrementally improves its estimation as soon as a new radar reading enters the system.

I will often reference the marvellous[1] book "Doing Bayesian Data Analysis" (Kruschke 2015), where most of my Bayesian knowledge comes from.

Motivation

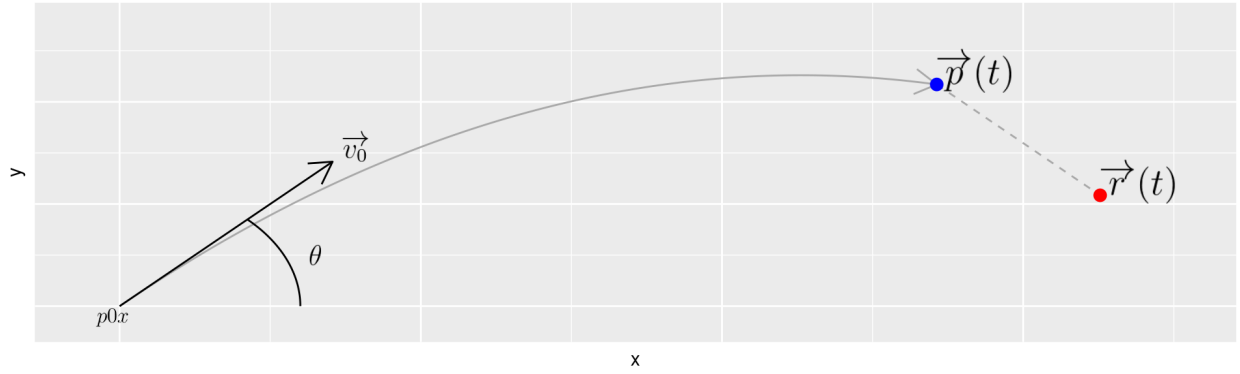
- to practice Bayesian analysis in a concrete and well-known setting
- to get a first impression of how it looks like to extend classic models (Physical, Electrical, Financial, etc) to incorporate domain (prior) knowledge
- for statistical fun
- for programming fun

not necessarily in order of importance.

[1] a joy to read - and with all five-stars reviews on Amazon, if you consider that almost all reviews with less then five-stars blame the Kindle formatting (that has issues with formulae), not the content at all

Physical model

The physical model is composed by a *deterministic* part (the trajectory) and a *statistical* one (the error characterization), that together describe the radar readings that we osberve:



The trajectory is described by the position vector $\overrightarrow{p}(t)$, that must satisfy the usual differential equation $\ddot{\overrightarrow{p}}(t) = \overrightarrow{g}$, where $\overrightarrow{g}$ is the gravitational acceleration. It is well-known that $\overrightarrow{p}(t)$ is perfectly and deterministically dictated by the initial state of the system (position and velocity when the battery fired at time t_0), i.e. by this quartet of four parameters:

- p_{0x}, the sought-after position of the battery, i.e. $\overrightarrow{p}(t_o)_x$
- v_0, the modulus of the initial velocity $\overrightarrow{v_0}$, aka "muzzle velocity"
- θ, the barrel elevation angle
- t_0, the time when the battery fired.

The radar readings are simply $\overrightarrow{r}(t_i) = \overrightarrow{p}(t_i) + \overrightarrow{e}(t_i)$, that is, the trajectory sampled by the radar at some instants t_i plus some reading error $\overrightarrow{e}$ (aka "measurement error").

The radar was built by our military-industrial complex, and hence we know everything about it: our ACME-MARK3 radar samples the sky using a constant sampling period (picture a quickly rotating radar as seen in movies...) and we know the stochastic characteristics of its error $\overrightarrow{e}$. To keep things simple[2], we will make the usual toy assumptions that $\overrightarrow{e}$ samples are i.i.e. (Indipendent and Identically Distributed) as a normal distribution, with mean zero (otherwise it would not be an error) and perfectly known covariance matrix, with indipendent components $\overrightarrow{e}_x$ and $\overrightarrow{e}_y$ both sharing the same value σ_e of the standard deviation[3].

We have enough information to simulate the stochastic process:

```
simulate <- function(n,          # number of readings
                     t0,         # sec
                     p0x,        # m
                     v0,         # m/s
                     theta,      # rad
                     error.sd,   # m
                     time.interval = c(0, 1) # as fraction of flight time
                    )
{
```

[2] in a real setting, we should observe that (for a classic rotating radar) the distance is estimated by the time the radiowave takes to bounch off the projectile, and the direction by the location of the intensity peek of the received radio echo during the radar rotation. Both are two very different physical processes (the former electrical, the latter mechanical), maybe approximately Gaussian, maybe indipendent, but with different standard deviations for sure; we should therefore at least vary the covariance matrix depending on the polar coordinates of the trajectory with respect to the radar location. Very fun, but I want to focus on the Bayesian part, not on the model adherence to reality.

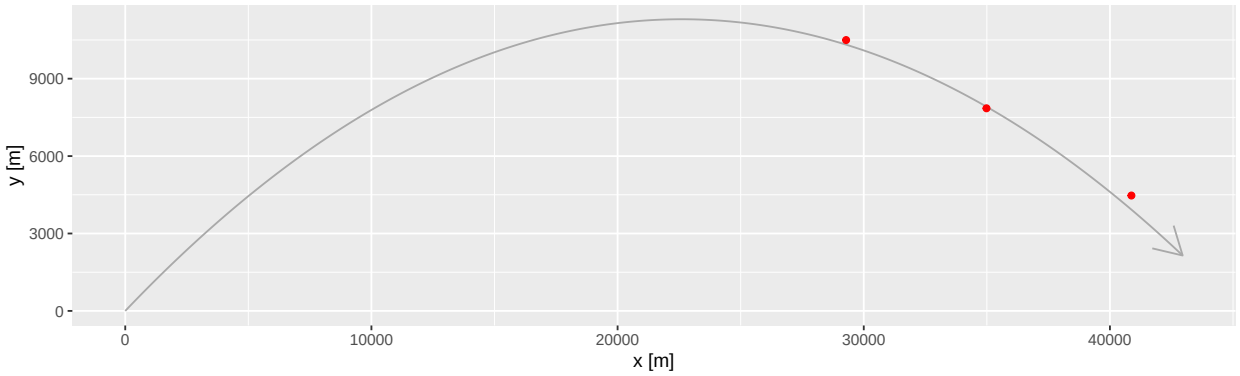
[3] i.e. the covariance matrix is

$$\Sigma = \begin{bmatrix} \sigma_e^2 & 0 \\ 0 & \sigma_e^2 \end{bmatrix}$$

```
  v0x <- v0 * cos(theta)
  v0y <- v0 * sin(theta)

  # calculate final impact figures
  flight_time <- 2 * v0y / g
  tf  <- t0 + flight_time
  pfx <- p0x + v0x * flight_time

  # sample time at uniform intervals
  t <- seq(t0 + time.interval[[1]]*flight_time,
           t0 + time.interval[[2]]*flight_time,
           length.out = n)

  # sample trajectory
  px <- p0x + v0x * (t - t0)
  py <-   0 + v0y * (t - t0) - 0.5 * g * (t - t0)^2

  # add radar error
  rx <- px + rnorm(n, 0, error.sd)
  ry <- py + rnorm(n, 0, error.sd)

  tibble(i=1:n, t=t, rx=rx, ry=ry)
}
```

And we observe this single shot of the battery, that we will use from now on as the input to our algorithm:

```
# radar readings (samples of trajectory, plus reading error)
set.seed(4321)
readings <- simulate(readings.n,
                     t0=0, p0x=p0x.true, v0=v0.true, theta=theta.true,
                     error.sd=readings.error.sd,
                     time.interval = c(0.65, 0.9))

# true trajectory
trajectory <- simulate(300,
                       t0=0, p0x=p0x.true, v0=v0.true, theta=theta.true,
                       error.sd=0,
                       time.interval = c(0, 0.95))

ggplot() +
  geom_line(data=trajectory, aes(x=rx,y=ry), arrow=grid::arrow(), color="darkgray") +
  geom_point(data=readings, aes(x=rx, y=ry), color="red") +
  xlab("x [m]") + ylab("y [m]")
```



i	t	rx	ry
1	62.40704	29282.85	10496.701
2	74.40839	34985.47	7852.220
3	86.40975	40872.61	4471.658

We can use a mere three readings at the end of the trajectory because our radar, located near the battery's target, can observe just a narrow region of the sky next to it.

Bayesian model

Here is the JAGS Bayesian model (illustrated below):

```
jagsModel <- "
model {
  for ( i in 1:N ) {
    ## radar readings (likelihood)
    rx[i] ~ dnorm( px[i], 1/readings.error.sd^2 )
    ry[i] ~ dnorm( py[i], 1/readings.error.sd^2 )

    ## positions
    px[i] <- p0x + v0x * (t[i] - t0)
    py[i] <-   0 + v0y * (t[i] - t0) - 0.5 * g * (t[i] - t0)^2
  }

  ## v0 components
  v0x <- v0 * cos(theta)
  v0y <- v0 * sin(theta)

  # priors
  p0x   ~ dnorm( priorP0xMean, 1/priorP0xSd^2 ) # see discussion
  v0    ~ dunif( 1 * 343, 3 * 343 )  # between MACH1 and MACH3
  theta ~ dunif( 0, pi/2 )           # aiming above ground and not backwards
  t0    ~ dunif( t[1] - 1000, t[1] ) # at most 1000 seconds before first radar reading time
}
"
```

The physical model already implicitly specified the Likelihood, that is, the probability of observing the data (the three $\overrightarrow{r}(t_i)$ and their corresponding t_i) given the four parameters. The formula[4] is simple and can

[4] $\Pr(D \mid p_{0x}, v_0, \theta, t_0) = \prod_{i=1}^{3} \{\mathcal{N}(\overrightarrow{r}(t_i) - \overrightarrow{p}(t_i), \Sigma)$; Σ as before; actually a density, not a probability

be easily reverse-engineered from the JAGS model text; I like to note that it can be mentally constructed by "plugging the four numbers" into the battery, firing the projectile and then following (simulating[5]) the projectile as it travels across the sky, writing down its positions at the three t_i instants, substracting the actual readings, calculating the three values of the Gaussian PDFs, and eventually multiplying.

Please also note that the JAGS model is essentially a *copy-and-paste of the physical model simulator above*, plus the priors. That's the beauty of the Bayesian approach that Laplace has given us: we just need to reason about how to generate (describe) the data given the parameters (which is relatively easy), and then have the computer turn it around from the data to the parameters[6].

Side note: ince almost all classical Physical models (in Mechanics, Thermodinamics, Electrical Engineering, etc etc) are composed by a deterministic part (like our trajectory) plus a measurement error (usually Gaussian) that produces the data (like our radar readings), from which the Likelihood functions is readily obtained, that means that we can leverage the sophisticated models produced by hundreds of years of scientific/engineering results and rather easily extend them to incorporate any other knowledge about the parameters (the priors). Isn't this great?

choice of priors

We will choose all mildly informative priors, obviously with the exception of the battery position.

battery position p_{0x}

Our Intelligence Department has sent us a note:

"The battery is located at x=100 meters, plus/minus 800 meters."

Turning this statement into a mathematical expression is one of the hardest part in Bayesian analysis, and depends on how much the Intelligence Department is credible (Are they trustable? Are they precise or sloppy? Have they provided good estimates in the past?) and on their culture and language as well (What they mean by "plus/minus"?). And this choice is critical, since a wrongly informative prior can badly influence the posterior, and a well-constructed one, at the opposite, can greatly enhance our chances of success. We hence performed a thorought investigation and assesment about the statement.

We will compare two possible outcomes:

- "Intelligent Intelligence": they are smart and cautious, their plus/minus represents their uncertainty well, and they do not exclude that the battery is outside that range. We will hence model the prior as a Gaussian with mean 100m and standard deviation 800m;
- "Uninformative": their estimate is junk, we are better off ignoring them. A good uninformative prior would be an uniform distribution covering the whole battlefield; but for simplicity we will use a Gaussian with a gigantic standard deviation (to keep the JAGS model code simple; I have experimented with an uniform as well and the results are qualitatively the same).

Spoiler: the true battery position is at x=0; the first choice would be the best.

initial velocity v_0

We don't know the battery model, and hence we will assume the typical range of a common howitzer: between MACH1[7] and MACH3. We have no preference for any velocity, and hence we will use an uniform distribution.

[5] we can actually *calculate* the trajectory since we were able to obtain the trajectory as a closed form solution. If we were to use a more realistic physical model (that e.g. considers also the air friction, the different air density at various altitudes etc) we would need to turn to *numerical simulation*, following the projectile using a program.

[6] that's the reason why the Bayesian approach is also known as "inverse probability"

[7] speed of sound of air near the battery location; assumed as 343 m/s (normal conditions: sea level, 20°C, etc)

barrel elevation θ

No preference either for the elevation; we will assume a uniform distribution between 0 and $\pi/2$ rad (that is, the battery does not aim underground and aims to the right).

firing time t_0

We will very conservatively assume that the battery fired at most 1000 seconds before the first radar reading, with no preference for any time (uniform distribution). This setting has a negligible influence on the posterior.

MCMC by JAGS

Here is the R function to drive JAGS:

```
# Run model and sample either the prior or the posterior
# distribution according to parameter `sampledDistribution`
runModelCore <- function(readings,
                         sampledDistribution = "posterior",
                         priorP0xMean,
                         priorP0xSd)
{
  # prepare input informations to Jags
  dataList <- list(
    t            = readings$t,
    rx           = readings$rx,
    ry           = readings$ry,
    N            = nrow(readings),
    priorP0xMean = priorP0xMean,
    priorP0xSd   = priorP0xSd,
    readings.error.sd = readings.error.sd,
    g = g,
    pi = pi
  )

  # if requested, sample the prior by removing the data samples, but not the structure
  # check [Kruschke, 2015], page 211
  if (sampledDistribution == "prior") {
    dataList$rx <- NULL
    dataList$ry <- NULL
  }

  # JAGS reads the model specification from the filesystem
  writeLines( jagsModel, con="TEMPmodel.txt" )

  # init chains with the true values to speed up convergence.
  # Of course in a real application we would either use MLE estimators
  # or let JAGS figure out the initial values itself, since the true values
  # would not be available.
  # RND seeds set for reproducibility only (it is not usually necessary, actually harmful)
  initsList <- list(
    list (p0x = p0x.true, v0=v0.true, theta=theta.true, t0=0,
          .RNG.name="base::Wichmann-Hill", .RNG.seed=31),
    list (p0x = p0x.true, v0=v0.true, theta=theta.true, t0=0,
          .RNG.name="base::Marsaglia-Multicarry", .RNG.seed=13),
    list (p0x = p0x.true, v0=v0.true, theta=theta.true, t0=0,
```

```
          .RNG.name="base::Super-Duper", .RNG.seed=17))

  # init and run samplers adaptation phase
  n.chains=3
  jagsState <- jags.model( file="TEMPmodel.txt", data=dataList, inits=initsList,
                           n.chains=n.chains,
                           n.adapt=500 # samplers adaptation (not a Markov Chain)
                         )
  unlink("TEMPmodel.txt")

  # burn in
  n.iter.burn.in <- 100001
  update( jagsState, n.iter=n.iter.burn.in, by = n.iter.burn.in / 4 )

  # sample the distribution
  # note: a lot of samples are needed for HDI accuracy,
  #       since the chains are highly correlated
  n.iter.output <- 100000*6 # a lot of samples needed for HDI accuracy
  n.thin.steps = 1
  coda.samples( jagsState,
                variable.names=c("p0x", "v0", "theta", "t0"),
                n.iter=ceiling(n.iter.output*n.thin.steps/n.chains),
                thin=n.thin.steps,
                by= n.iter.output*n.thin.steps / 4
              )
}
```

The function is pretty standard and hence we won't illustrate it; we refer the interested reader to the JAGS manual (Plummer 2017) or for a great detailed explanation to (Kruschke 2015, especially since page 195), where I got the early template from. Just note that we can use it to sample the prior as well as the posterior.

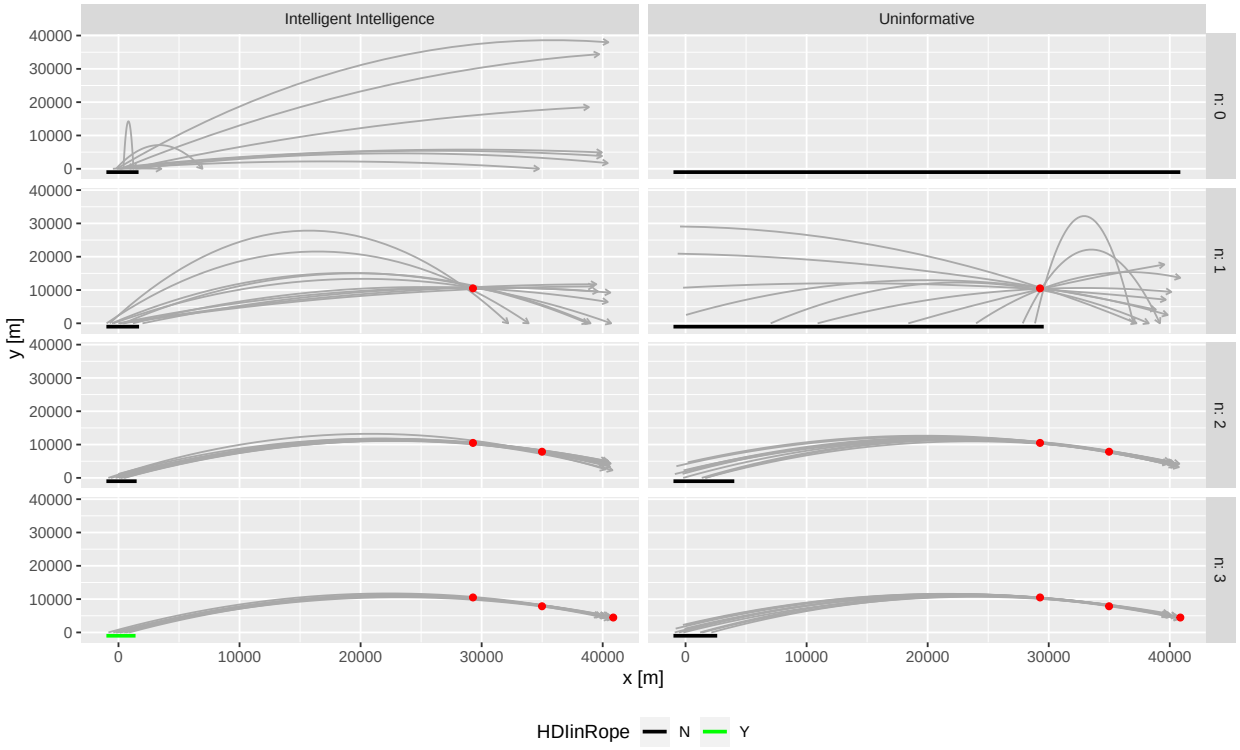
The chains converge and are accurate enough for our purposes; we will briefly illustrate them at the end.

Bayesian Updating Progress

We can now perform the most interesting part of the discussion: explore how each radar reading improves our knowledge about the parameters, from the initial prior to the final posterior. Letting n be the number of radar readings received so far, we will observe how our estimate of $p0x$ progresses from n = 0 (as dictated by the prior) to n=1 (after the first reading), all along to n=3.

Here is the overall result[8] (we will discuss each step in detail later):

[8] the drawing code is a rather technical mix of CODA wrangling and ggplot2/tidyverse magics. It can be read in the rmarkdown source at the github repository quoted at the bottom



Each grey line is a credible trajectory: we sample 10 quartets at random from the posterior chain (actually the prior for n=0) and draw the corresponding trajectory. This is a wonderful way to visualize what the Bayesian procedure is, so to speak, "thinking" and "believing".

The red dots are the radar readings used to compute the posterior; it is manifest that the credible trajectories pass "near"[9] them, as intuitively expected.

The trajectories originate from the credible $p0x$ positions; underneath them the 95% HDI[10] of $p0x$ is drawn, that can be intuitively described as the region the contains the 95% most credible positions of the battery.

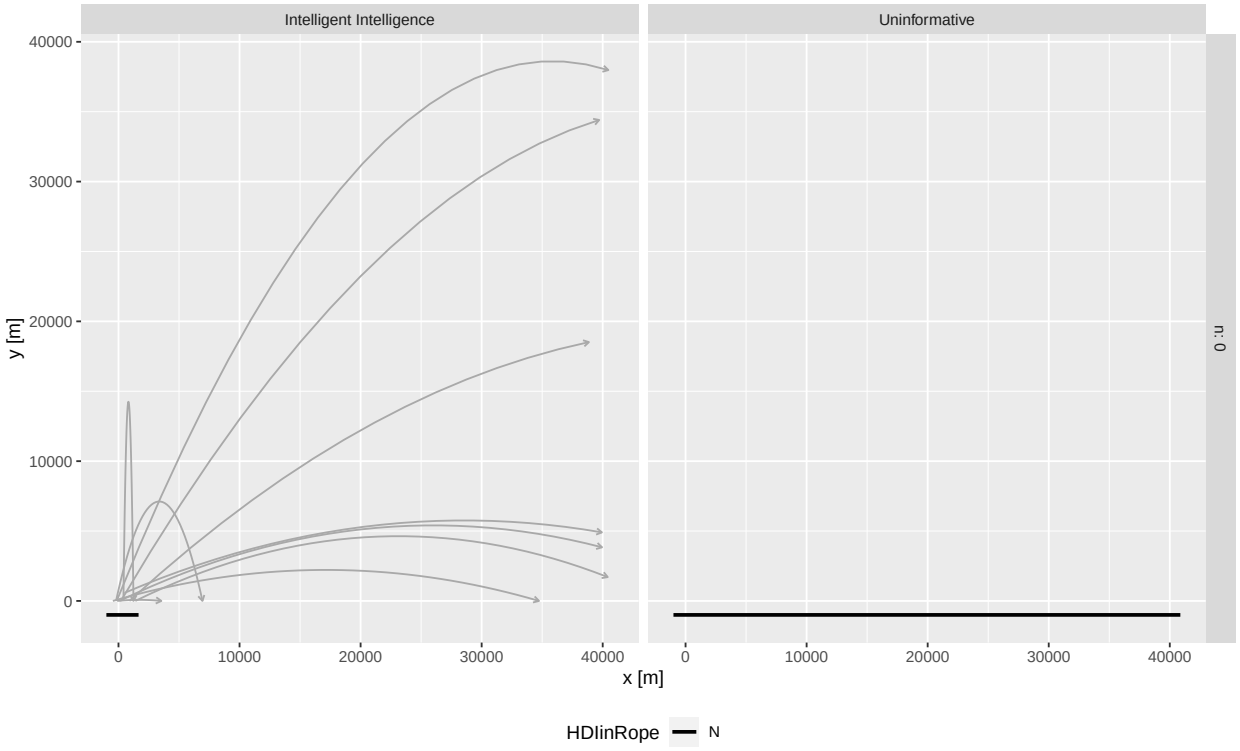
The HDI turns green when it is completely contained inside the ROPE, the Region Of Practical Equivalence (Kruschke 2015, definition at page 336), that we might describe as the sets of points *practically* equivalent to the true position of the battery. "Equivalent" is defined by the *practical* purpose of our analysis: in this case we will assume that we plan to send a drone to confirm the position of the battery, and since our drone can explore a space of 1500m from the starting point (in both directions) before running out of power, the ROPE interval is centered at the true position and 2x1500m wide, because any starting point inside this ROPE will *equivalently* allow us to find the battery (one or another, we don't care). When the 95% HDI is inside the ROPE, to the best of our knowledge (prior + data) the drone will find the battery with (at least) 95% probability.

Why 95% and not, say, 40%? In a real scenario we would compare the value gained by finding the battery with the cost of the drone mission, and decide a threshold accordingly. A 95% threshold is compatible with a very high cost compared to the gain: we want to be almost absolutely sure the the drone is not wasted. If the drone were inexpensive, we might choose a very low threshold instead, maybe even 10%.

n = 0: before the first reading

Let's zoom in to the priors:

[9] The smaller the radar σ_e, the closest they pass near the readings (not shown here but easy to check)
[10] Highest Density Interval, again check (Kruschke 2015, definition at page 28 and used all around the book)

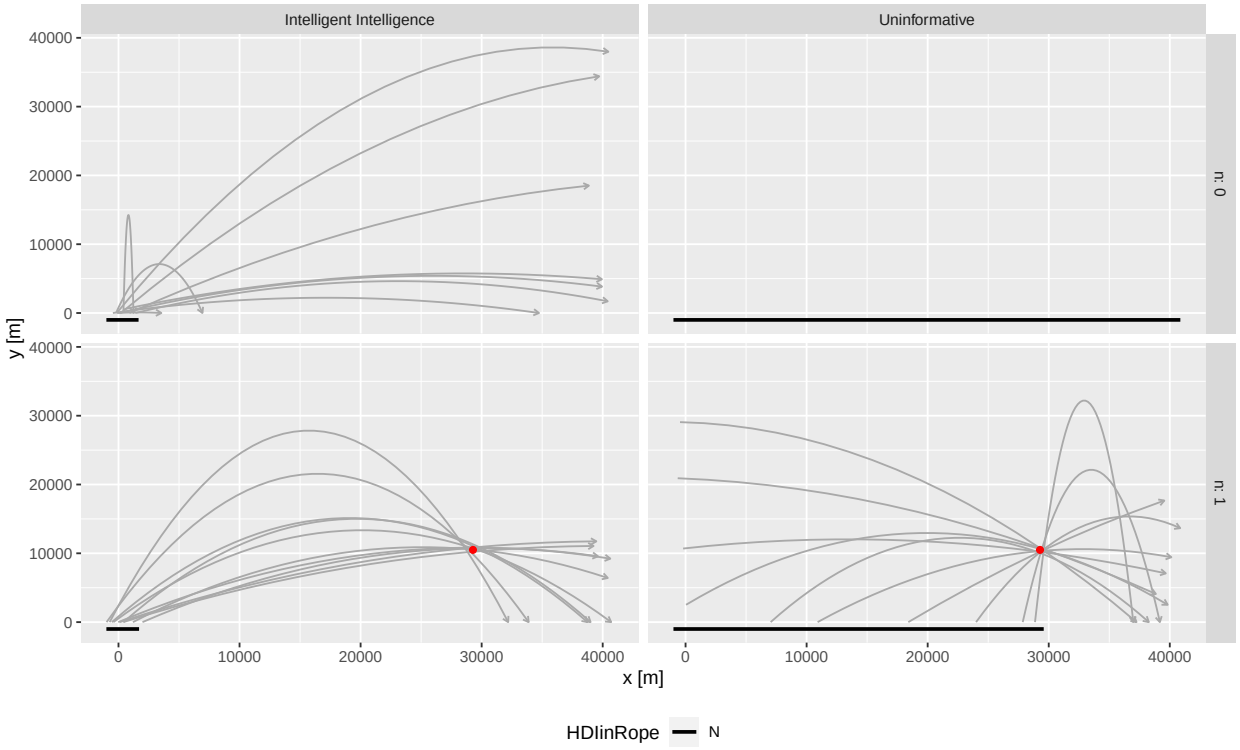


The "Intelligent Intelligence" variant shoots credible projectiles mostly from the HDI, but then the trajectories diverge quite wildly. They do not diverge completely randomly, however, and exhibit quite a strong regularity due to the other priors. In fact, the θ prior forces them all to point to the right (since the prior has exactly zero density above $\pi/2$). And since $v0$ is constrained below MACH3, the kinetic energy is constrained as well, and that caps both the maximun altitude and, indirectly, the maximum range in the x direction. As a result, the credible trajectories do behave like a weakly-coordinated flock.

The "Uninformative" variant shoots credible projectiles from random points[11] in the battlefield, and hence, even it shares the same other priors with the former, it looks almost random. In fact the chances of passing inside our narrow window over the battlefield are almost zero, and the right window very frequently contains only a few trajectory fragments if any at all (depending on the seeds set for JAGS/R pseudorandom generators). Their credible trajectory shape is, however, the same as the other variant.

[11] the 95% HDI extends well beyond the window shown - its width is actually a gigantic 4×10^6m

n = 1: after the first reading



As the first reading arrives, the credible trajectories cluster together near the reading[12], since the Likelihood function value peaks around it: the posterior peaks as well and, as a consequence, shrinks everywhere else (the "Reallocation of Credibility"[13] of Bayesian Update).

It's instructive to describe how we could manually draw the approximate credible trajectories using only elementary Statistics, Math and Physics. Let's focus on the bottom left window. We could recall from high-school that the trajectories are parabolic and hence with three degrees of freedom (the three parameters in $y = \alpha_0 + \alpha_1 x + \alpha_2 x^2$), and by varing the three parameters we can position the parabola on the window wherever we want. We choose a starting point inside the HDI and force the parabola to pass there; this still leaves us two degrees of freedom. Then, we draw a small circle centered on the reading with radius $3 * \sigma_e$: with 99% probability the true trajectory passed through our circle, so we choose another point there, and nail the parabola there as well.

We have still a degree of freedom, that allows the trajectory to vary wildly (the projectile could for example get to the Moon[14] and back). Only by factoring in more prior knowledge we might get to a family of trajectories roughly comparable to the Bayesian one; for example we could constrain the maximum height of the parabola by recalling that the initial kinetic energy is constrained by v_0, that is between MACH1 and MACH3 (doable but laborious).

Equivalently: choose a starting point inside the HDI, aim at the circle, and fire. We can roughly factor in the prior on $v0$ by choosing different models of howitzers with different initial velocity, all between MACH1 and MACH3.

In a way, the Bayesian algorithm does the same reasoning in this scenario, but of course with much greater

[12] because σ_e is relatively small compared to the trajectory

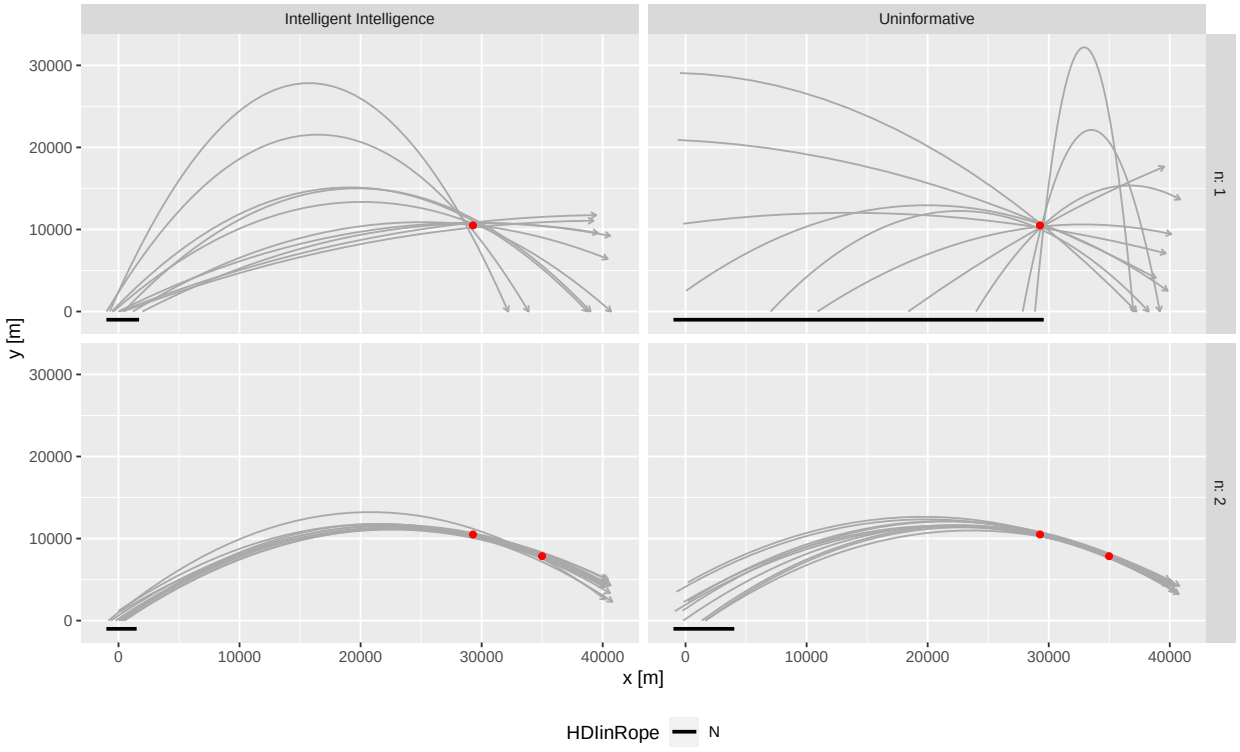
[13] i.e. check (Kruschke 2015), especially figure 2.1 on page 17, the best illustration I have ever seen

[14] until now we have only factored-in the fact about the trajectory shape (parabolic), nothing has been said about the initial velocity, starting time, or travel time. It might as well travel to Alpha Centauri an back in 1 millisecond, since the model is classic and not relativistic, and hence the light-speed limit is not in effect.

precision, sound statistical background and less manual work from our part (or at least less boring).

As a final observation, note how this first reading has not altered that much the HDI on the left window; the additional information has mainly shrunken the credible trajectories (loosely speaking: by reducing their degrees of freedom). The "Uninformative" scenario HDI is very different instead: for example now it lays on the left of the reading, thanks to the prior on θ, while the credible trajectories are still quite free.

n = 2: after the second reading

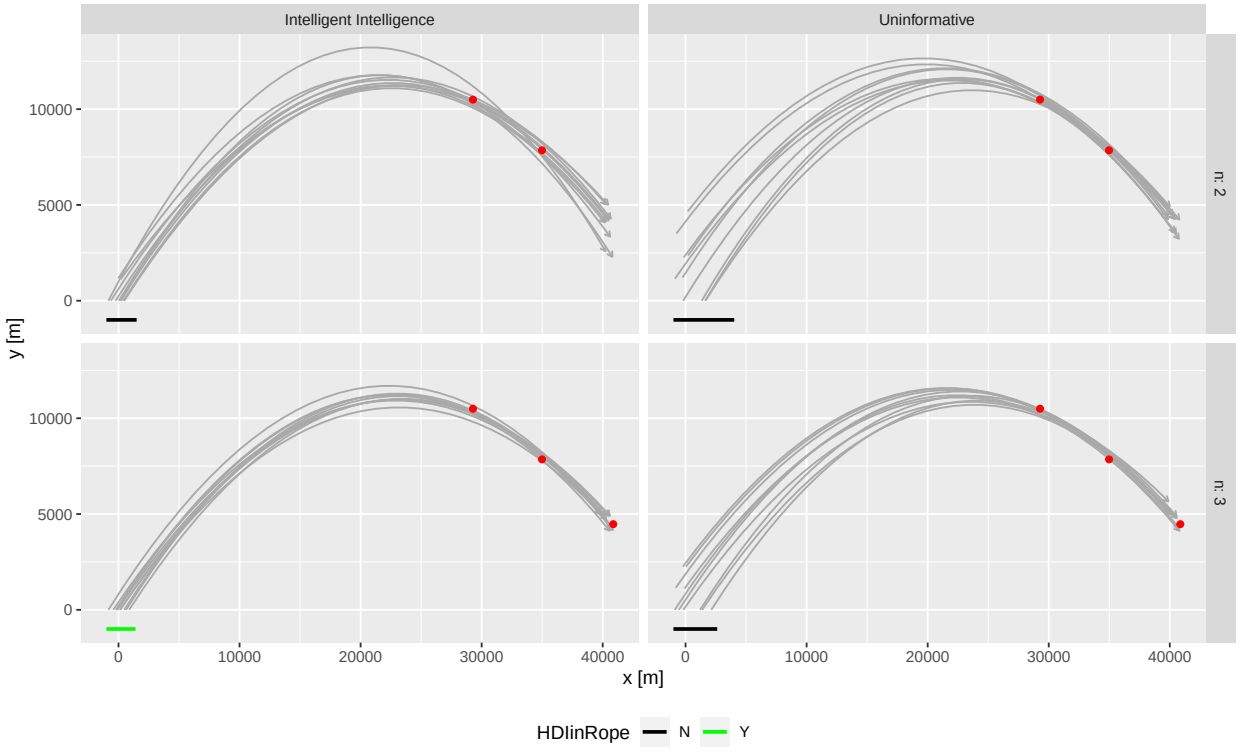


Now with two readings, the credible trajectories gets clustered much more around the true trajectory (that starts at x=0).

Using the manual methods, we would have now two points to nail the parabola to (or to aim to), and a smaller room to start the trajectory, and hence much less freedom for our projectile.

Again, the HDI shrinks much more for the "Uninformative" scenario on the right.

n = 3: after the third reading

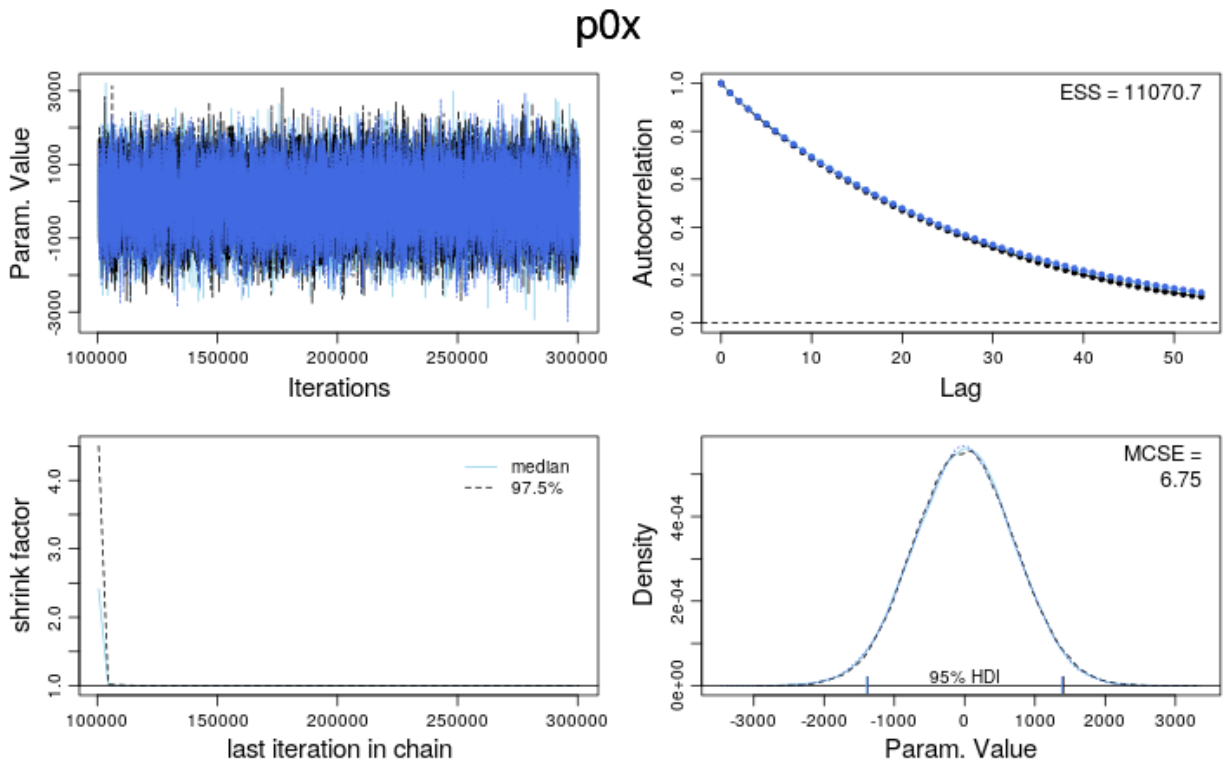


At this point, if the three readings were noiseless, we could simply fit the parabola to them and we would have inferred the exact position of the battery, making the priors useless. But since σ_e is small but not zero, there's still some room for the parabola to move around, and hence opportunity for the priors to provide useful information by influencing the prosterior; and in fact "Intelligent Intelligence" sets the green light to launch the drone, but that's not the case for "Uninformative".

Chains Convergence

We now breafly look at the converge diagnostics and accuracy of the chains. We will look at $p0x$, but I have checked all the other three chain parameters as well and the results are the same, unless otherwise noted.

Here are the four diagnostics diagrams suggested by (Kruschke 2015, especially chapter 7.5 on page 178) and implemented in his diagMCMC() function:



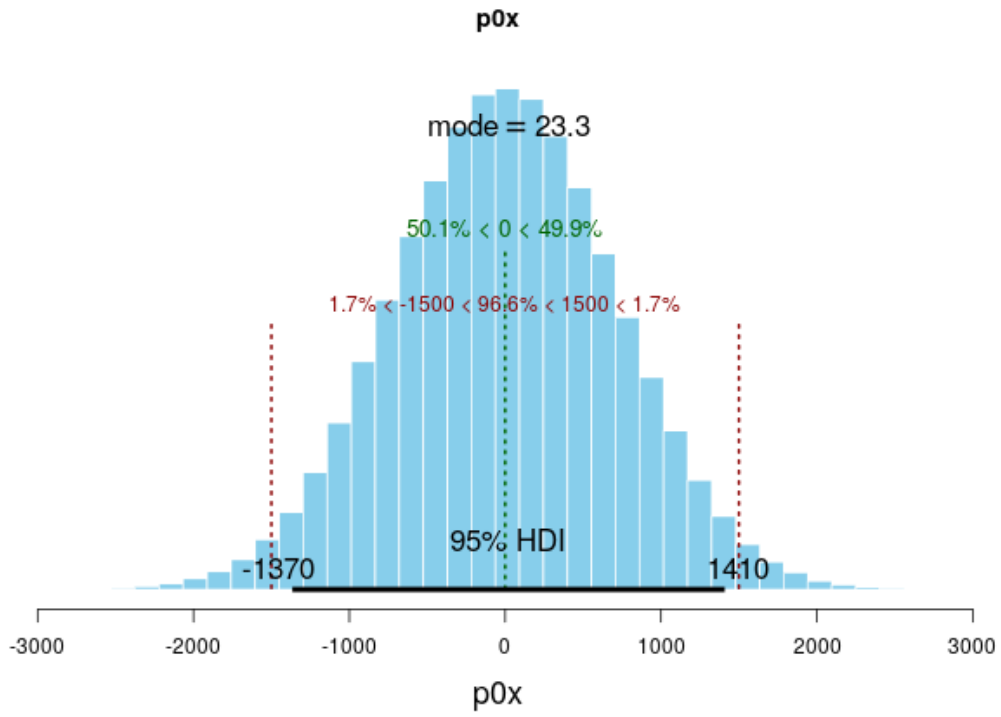
We can see that, from the top left panel counterclockwise:

- the three chain jumps are qualitatively the same, with no orphans
- the Gelman-Rubin statistic is well below 1.1
- the three chains share the same estimated marginal density
- the autocorrelation is high, but I have got enough samples to get over the minimum ESS (Effective Sample Size) of 10,000 that is empirically[15] needed to get good HDI accuracy and stability.

The other three parameters diagnostics (not shown) behave the same, but the ESS is much lower (and this is not an issue since we are not interested in their HDIs).

And of course - let's end with a souvenir photo of the longed-for $p0x$ posterior, showing the ROPE and the $p0x$ true value:

[15] from (Kruschke 2015, 184): "For reasonably accurate and stable estimates of the limits of the 95% HDI, an ESS of 10,000 is recommended. This is merely a heuristic based on experience with practical applications"



We've found the battery!!

Next steps

Some ideas (and random thoughts) that I wrote down while I was working.

air friction and simulation

The next most natural steps would be to consider air friction.

Closed-form solutions exist for special cases (e.g. for air friction force modulus equal to $v^2(t)$), and for those it is probably simple to modify the likelihood accordingly. I would expect less precision in the final p_0x estimation since now many more trajectories can "pass near" the radar readings[16], that are located near the end of the trajectories; but it would be fun to check out.

For a more realistic model, we would need to turn to simulation. The real air friction dependency on $v(t)$ is complex, it varies depending on the speed of sound that in turn depends on air pressure and hence on $y(t)$, and it does not help that the projectile starts hypersonic and (usually) ends subsonic, thus breaking the sound barrier at MACH1.

How to integrate a simulation in JAGS? I've investigated a bit and discovered that JAGS could be extended not only by providing new probability distributions, but also new functions. JAGS is written in C++ and thus we could write the simulation in that language, incorporate it in the JAGS source (it's open-source) and recompile; or maybe provide the function as a shared library and relink.

[16] the models with air friction tends to produce trajectories that are more vertical near the end (since they admit a vertical asymptote), hence there are more credible trajectories compatible with σ_e - imagine a basketball hoop. Equivalently: if you integrate the model backwards in time, a small difference in the initial codition, compatible with σ_e, makes a more different trajectory.

Stan

Obviously, checking STAN would be very easy; I would expect similar results, but maybe faster runtimes and less autocorrelation on the posterior chains, since I suspect (by intuition only, I haven't checked) that the posterior is constrained in the "narrow valleys" described in (Kruschke 2015, especially page 176). My intuition should be verified of course.

Nimble

It would be very interesting to explore the Nimble tool (https://r-nimble.org/), that extends the BUGS language (the language of the JAGS models) to allow for easy integration of new functions, written in simil-R, and compiles them in C++ before running. On paper, it has the potential for being a fantastic toolset, worth investing time into.

The simple frictionless model that we have discussed looks like a good testcase to compare Nimble with JAGS (and maybe STAN), before turning to simulation with air friction where it should show its full potential.

Document version: 2020/05/18

References

Gihub repository: (https://github.com/alberto-dellera/bayesian_artillery)

Kruschke, John K. 2015. *Doing Bayesian Data Analysis: A Tutorial with R, Jags, and Stan.* 2nd ed. Academic Press / Elsevier. https://sites.google.com/site/doingbayesiandataanalysis.

Plummer, Martyn. 2017. *JAGS Manuals.* https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/.