Markov Chain Monte Carlo for Inverse Problems

David Ochsner

May 22, 2020

Contents

1 Theory

1.1 Papers

1.1.1 Stuart et al: Inverse Problems: A Bayesian Perspective [3]

Theoretical Background [1]

Notation Central equation:

$$y = \mathcal{G}(u) + \eta$$

with:

- $y \in \mathbb{R}^q$: data
- $u \in \mathbb{R}^n$: IC ("input to mathematical model")
- $\mathcal{G}(\cdot) : \mathbb{R}^n \to \mathbb{R}^q$: observation operator
- η: mean zero RV, observational noise (a.s. $\eta \sim \mathcal{N}(0, \mathcal{C})$)

1.1.2 Cotter et al: MCMC for functions [1]

Implementation, MCMC in infinite dimensions

[1] this is a test footnote

1.1.3 Schneider et al: Earth System Modeling 2.0 [2]

Example for MCMC on ODE

1.2 Small results

1.2.1 Gaussian in infinite dimensions

This section is quite a mess, maybe you could suggest a not-too-technical introduction to infinite dimensional Gaussian measures?

Wiki: Definition of Gaussian measure uses Lesbesgue measure. However, the Lesbesgue-Measure is not defined in an infinite-dimensional space (wiki).

Can still define a measure to be Gaussian if we demand all push-forward measures via a linear functional onto $\mathbb{R}$ to be a Gaussian. (What about the star (E^*, L_*) in the wiki-article? Are they dual-spaces?) (What would be an example of that? An example for a linear functional on an inf-dims space given on wikipedia is integration. What do we integrate? How does this lead to a Gaussian?)

How does this fit with the description in [1]? -> Karhunenen-Loéve

What would be an example of a covariance operator in infinite dimensions? The Laplace-Operator operates on functions, the eigenfunctions would be *sin*, *cos* (I think? This might not actually be so easy, see Dirichlet Eigenvalues). Are the eigenvalues square-summable?

Anyway, when a inf-dim Gaussian is given as a KL-Expansion, an example of a linear functional given as $f(u) = \langle \phi_i, u \rangle$ for ϕ_i an eigenfunction of $\mathcal{C}$, then I can see the push-forward definition of inf-dim Gaussians satisfied. ($\mathcal{C}$ spd, so ϕ_i s are orthogonal, so we just end up with one of the KH-"components" which is given to be $\mathcal{N}(0, 1)$).

The problem is not actually in $\exp\left(-1/2x^T\mathcal{C}^{-1}x\right)$. What about $\exp\left(-1/2\left\|\mathcal{C}^{-1/2}x\right\|\right)$?

What about the terminology in [1]? Absolutely continuous w.r.t a measure for example?

How is the square root of an operator defined? For matrices, there seems to be a freedom in choosing whether $A = BB$ or $A = BB^T$ for $B = A^{1/2}$. The latter definition seems to be more useful when working with Cholesky factorizations (cf. `https://math.stackexchange.com/questions/2767873/why-is-the-square-root-of-cholesky-decomposition-equal-to-the-lower-` but for example in the wiki-article about the matrix (operator) square root (`https://en.wikipedia.org/wiki/Square_root_of_a_matrix`): "The Cholesky factorization provides another particular example of square root, which should not be confused with the unique non-negative square root."

1.2.2 Bayes' Formula & Radon-Nikodym Derivative

Bayes' Formula is stated using the Radon-Nikodym Derivative in both [1] and [3]:

$$\frac{\mathrm{d}\mu}{\mathrm{d}\mu_0} \propto \mathrm{L}(u),$$

where $\mathrm{L}(u)$ is the likelihood.

Write the measures as $\mathrm{d}\mu = \rho(u)\mathrm{d}u$ and $\mathrm{d}\mu_0 = \rho_0(u)\mathrm{d}u$ with respect to the standard Lesbesgue measure. Then we have

$$\int f(u)\rho(u)\mathrm{d}u = \int f(u)\mathrm{d}\mu(u) = \int f(u)\frac{\mathrm{d}\mu(u)}{\mathrm{d}\mu_0(u)}\mathrm{d}\mu_0 = \int f(u)\frac{\mathrm{d}\mu(u)}{\mathrm{d}\mu_0(u)}\rho_0(u)\mathrm{d}u,$$

provided that $\mathrm{d}\mu$, $\mathrm{d}\mu_0$ and f are nice enough (which they are since we're working with Gaussians). This holds for all test functions f, so it must hold pointwise:

$$\frac{\mathrm{d}\mu(u)}{\mathrm{d}\mu_0(u)} = \frac{\rho(u)}{\rho_o(u)}.$$

Using this we recover the more familiar formulation of Bayes' formula:

$$\frac{\rho(u)}{\rho_o(u)} \propto \mathrm{L}(u).$$

1.2.3 Acceptance Probability for Metropolis-Hastings

A Markov process with transition probabilities $t(y|x)$ has a stationary distribution $\pi(x)$.

- The existence of $\pi(x)$ follows from *detailed balance*:

  $$\pi(x)t(y|x) = \pi(y)t(x|y).$$

  Detailed balance is sufficient but not necessary for the existence of a stationary distribution.

- Uniqueness of $\pi(x)$ follows from the Ergodicity of the Markov process. For a Markov processto be Ergodic it has to:
  - not return to the same state in a fixed interval
  - reach every state from every other state in finite time

The Metropolis-Hastings algorithm constructs transition probabilities $t(y|x)$ such that the two conditions above are satisfied and that $\pi(x) = P(x)$, where $P(x)$ is the distribution we want to sample from.

Rewrite detailed balance as

$$\frac{t(y|x)}{t(x|y)} = \frac{P(y)}{P(x)}.$$

Split up the transition probability into proposal $g(y|x)$ and acceptance $a(y, x)$. Then detailed balance requires

$$\frac{a(y,x)}{a(x,y)} = \frac{P(y)g(x|y)}{P(x)g(y|x)}.$$

Choose

$$a(y,x) = \min\left\{1, \frac{P(y)g(x|y)}{P(x)g(y|x)}\right\}$$

to ensure that detailed balance is always satisfied. Choose $g(y|x)$ such that ergodicity is fulfilled.

If the proposal is symmetric ($g(y|x) = g(x|y)$), then the acceptance takes the simpler form

$$a(y,x) = \min\left\{1, \frac{P(y)}{P(x)}\right\}. \tag{1}$$

Since the target distribution $P(x)$ only appears as a ratio, normalizing factors can be ignored.

1.2.4 Potential for Bayes'-MCMC when sampling from analytic distributions

How can we use formulations of Metropolis-Hastings-MCMC algorithms designed to sample from posteriors when want to sample from probability distribution with an easy analytical expression?

Algorithms for sampling from a posterior sample from

$$\rho(u) \propto \rho_0(u)\exp(-\Phi(u)),$$

where ρ_0 is the prior and $\exp(-\Phi(u))$ is the likelihood. Normally, we have an efficient way to compute the likelihood.

When we have an efficient way to compute the posterior ρ and we want to sample from it, the potential to do that is:

$$\Phi(u) = \ln(\rho_0(u)) - \ln(\rho(u)),$$

where an additive constant from the normalization was omitted since only potential differences are relevant.

When working with a Gaussian prior $\mathcal{N}(0, \mathcal{C})$, the potential takes the form

$$\Phi(u) = -\ln \rho(u) - \frac{1}{2}\left\|\mathcal{C}^{-1/2}u\right\|^2.$$

When inserting this into the acceptance probability for the standard random walk MCMC given in formula (1.2) in [1], the two Gaussian-expressions cancel, as do the logarithm and the exponentiation, leaving the simple acceptance described in 1.

This cancellation does not happen when using the pCN-Acceptance probablity. This could explain the poorer performance of pCN when directly sampling a probablity distribution.

1.2.5 Acceptance Probabilities for different MCMC Proposers

Start from Bayes' formula and rewrite the likelyhood $\mathrm{L}(u)$ as $\exp(-\Phi(u))$ for a positive scalar function Φ called the potential:

$$\frac{\rho(u)}{\rho_o(u)} \propto \exp(\Phi(u)).$$

Assuming our prior to be a Gaussian ($\mu_0 \sim \mathcal{N}(0, \mathcal{C})$).

Then

$$\rho(u) \propto \exp\left(-\Phi(u) + \frac{1}{2}\left\|C^{-1/2}u\right\|^2\right),$$

since $u^T C^{-1} u = (C^{-1/2}u)^T (C^{-1/2}u) = \langle C^{-1/2}u, C^{-1/2}u\rangle = \left\|C^{-1/2}u\right\|^2$, where in the first equality we used C being symmetric.

This is formula (1.2) in [1] and is used in the acceptance probability for the standard random walk (see also Acceptance Probability for Metropolis-Hastings)

$\mathcal{C}^{-1/2}u$ makes problems in infinite dimensions.

Todo: Why exactly is the second term (from the prior) cancelled when doing pCN?

1.2.6 Different formulations of multivariate Gaussians

Is an RV $\xi \sim \mathcal{N}(0, C)$ distributed the same as $C^{1/2}\xi_0$, with $\xi_0 \sim \mathcal{N}(0, \mathcal{I})$?

From wikipedia: Affine transformation $Y = c + BX$ for $X \sim \mathcal{N}(\mu, \Sigma)$ is also a Gaussian $Y \sim \mathcal{N}\left(c + B\mu, B\Sigma B^T\right)$. In our case $X \sim \mathcal{N}(0, \mathcal{I})$, so

$Y \sim \mathcal{N}\left(0, C^{1/2}\mathcal{I}C^{1/2^T}\right) = \mathcal{N}(0, C)$, since the covariance matrix is positive definite, which means it's square root is also positive definite and thus symmetric.

On second thought, it also follows straight from the definition:

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma) \Leftrightarrow \exists \mu \in \mathbb{R}^k, A \in \mathbb{R}^{k \times l} \text{ s.t. } \mathbf{X} = \mu + A\mathbf{Z} \text{ with } \mathbf{Z}_n \sim \mathcal{N}(0, 1) \text{ i.i.d}$$

where $\Sigma = AA^T$.

2 Framework/Package Structure

The framework is designed to support an easy use case:

```
proposer = StandardRWProposer(beta=0.25, dims=1)
accepter = AnalyticAccepter(my_distribution)
rng = np.random.default_rng(42)
sampler = MCMCSampler(rw_proposer, accepter, rng)

samples = sampler.run(x_0=0, n_samples=1000)
```

There is only one source of randomness, shared among all classes and supplied by the user. This facilitates reproducability.

Tests are done with `pytest`.

2.1 Distributions

A class for implementing probability distributions.

```
class DistributionBase(ABC):
    @abstractmethod
    def sample(self, rng):
        """Return a point sampled from this distribution"""
        ...
```

The most important realisation is the `GaussianDistribution`, used in the proposers.

```
class GaussianDistribution(DistributionBase):
    def __init__(self, mean=0, covariance=1):
        ...

    def sample(self, rng):
```

```python
        ...

    def apply_covariance(self, x):
        ...

    def apply_sqrt_covariance(self, x):
        ...

    def apply_precision(self, x):
        ...

    def apply_sqrt_precision(self, x):
        ...
```

The design of this class is based on the implementation in muq2. The `precision` / `sqrt_precision` is implemented through a Cholesky decomposition, computed in the constructor. This makes applying them pretty fast ($\mathcal{O}(n^2)$).

At the moment the there is one class for both scalar and multivariate Gaussians. This introduces some overhead as it has to work with both `float` and `np.array`. Maybe two seperate classes would be better.

Also, maybe there is a need to implement a Gaussian using the Karhunen-Loéve-Expansion?

2.2 Potentials

A class for implementing the potential resulting from rewriting the likelihood as

$$\mathrm{L}(u) = \exp(-\Phi(u)).$$

```python
class PotentialBase(ABC):
"""
    Potential used to express the likelihood;
    d mu(u; y) / d mu_0(u) \propto L(u; y)
    Write L(u; y) as exp(-potential(u; y))
    """
    @abstractmethod
    def __call__(self, u):
        ...
```

```
    @abstractmethod
    def exp_minus_potential(self, u):
        ...
```

The two functions return $\Phi(u)$ and $\exp(-\Phi(u))$ respectively. Depending on the concrete potential, one or the other is easier to compute.

Potentials are used in the accepters to decide the relative weight of different configurations. There, the `PotentialBase.exp_minus_potential` is used.

2.2.1 AnalyticPotential

This potential is used when sampling from an analytically computable probability distribution, i.e. a known posterior. In this case

$$\exp(-\Phi(u)) = \frac{\rho(u)}{\rho_0(u)},$$

see `theory.org`

2.2.2 EvolutionPotential

This potential results when sampling from the model-equation

$$y = \mathcal{G}\,(u) + \eta,$$

with $\eta \sim \rho$. The resulting potential can be computed as

$$\exp(-\Phi(u)) = \rho(y - \mathcal{G}\,(u)).$$

2.3 Proposers

Propose a new state v based on the current one u.

```
class ProposerBase(ABC):
    @abstractmethod
    def __call__(self, u, rng):
        ...
```

2.3.1 StandardRWProposer

Propose a new state as

$$v = u + \sqrt{2\delta}\xi,$$

with either $\xi \sim \mathcal{N}(0, \mathcal{I})$ or $\xi \sim \mathcal{N}(0, \mathcal{C})$ (see section 4.2 in [1]).

This leads to a well-defined algorithm in finite dimensions. This is not the case when working on functions (as described in section 6.3 in [1])

2.3.2 pCNProposer

Propose a new state as

$$v = \sqrt{1-\beta^2}u + \beta\xi,$$

with $\xi \sim \mathcal{N}(0, \mathcal{C})$ and $\beta = \frac{8\delta}{(2+\delta)^2} \in [0, 1]$ (see formula (4.8) in [1]).

This approach leads to an improved algorithm (quicker decorrelation in finite dimensions, nicer properties for infinite dimensions)(see sections 6.2 + 6.3 in [1]).

The wikipedia-article on the Cholesky-factorization mentions the use-case of obtaining a correlated sample from an uncorrelated one by the Cholesky-factor. This is not implemented here.

2.4 Accepters

Given a current state u and a proposed state v, decide if the new state is accepted or rejected.

For sampling from a distribution $P(x)$, the acceptance probability for a symmetric proposal is $a = \min\{1, \frac{P(v)}{P(u)}\}$ (see `theory.org`)

```
class ProbabilisticAccepter(AccepterBase):
    def __call__(self, u, v, rng):
        """Return True if v is accepted"""
        a = self.accept_probability(u, v)
        return a > rng.random()

    @abstractmethod
    def accept_probability(self, u, v):
        ...
```

2.4.1 AnalyticAccepter

Used when there is an analytic expression of the desired distribution.

```python
class AnalyticAccepter(ProbabilisticAccepter):
    def accept_probability(self, u, v):
        return self.rho(v) / self.rho(u)
```

2.4.2 StandardRWAccepter

Based on formula (1.2) in [1]:

$$a = \min\{1, \exp(I(u) - I(v))\},$$

with

$$I(u) = \Phi(u) + \frac{1}{2}\left\|\mathcal{C}^{-1/2}u\right\|^2$$

.

See also `theory.org`.

2.4.3 pCNAccepter

Works together with the pCNProposer to achieve the simpler expression for the acceptance

$$a = \min\{1, \exp(\Phi(u) - \Phi(v))\}.$$

2.4.4 CountedAccepter

Stores and forwards calls to an "actual" accepter. Counts calls and accepts and is used for calculating the acceptance ratio.

2.5 Sampler

The structure of the sampler is quite simple, since it can rely heavily on the functionality provided by the Proposers and Accepters.

```python
class MCMCSampler:
    def __init__(self, proposal, acceptance, rng):
        ...

    def run(self, u_0, n_samples, burn_in=1000, sample_interval=200):
        ...

    def _step(self, u, rng):
        ...
```

3 Results

3.1 Analytic sampling from a bimodal Gaussian

3.1.1 Setup

Attempting to recreate the "Computational Illustration" from [1]. They use, among other algorithms, pCN to sample from a 1-D bimodal Gaussian

$$\rho \propto (\mathcal{N}(3,1) + \mathcal{N}(-3,1))\mathbb{1}_{[-10,10]}.$$

Since the density estimation framework for a known distribution is not quite clear to me from the paper, I don't expect to perfectly replicate their results.

They use a formulation of the prior based on the Karhunen-Loéve Expansion that doesn't make sense to me in the 1-D setting (how do I sum infinite eigenfunctions of a scalar?).

The potential for density estimation described in section is also not clear to me (maybe for a similar reason? What is u in the density estimate case?).

I ended up using a normal $\mathcal{N}(0,1)$ as a prior and the potential described before, and compared the following samplers:

- (1) `StandardRWProposer` ($\delta = 0.25$) + `AnalyticAccepter`
- (2) `StandardRWProposer` ($\delta = 0.25$) + `StandardRWAccepter`
- (3) `pCNProposer` ($\beta = 0.25$) + `pCNAccepter`

The code is in `analytic.py`.

3.1.2 Result

All three samplers are able to reproduce the target density 1

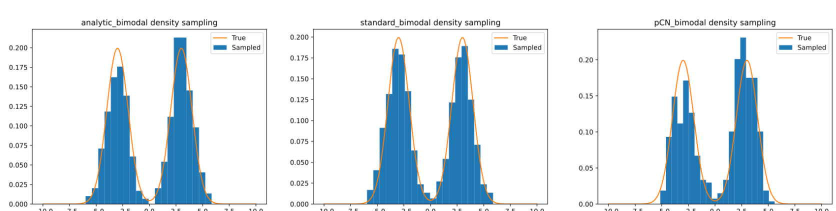

Figure 1: Burn-in: 1000, sample-interval: 200, samples: 500

The autocorrelation decays for all samplers: 2. However, the pCN doens't do nearly as well as expected. This could be the consequence of the awkward formulation of the potential or a bad prior.

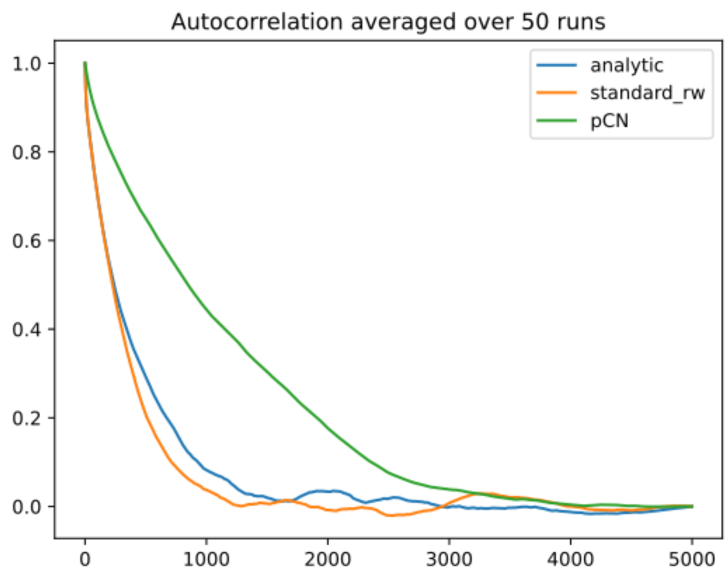

Figure 2: AC of bimodal distribution. pCN takes forever to decorrelate

3.2 Bayesian inverse problem for $\mathcal{G}(u) = \langle g, u \rangle$

For $\mathcal{G}(u) = \langle g, u \rangle$ the resulting posterior under a Gaussian prior is again a Gaussian. The model equation is

$$y = \mathcal{G}(u) + \eta$$

with:

- $y \in \mathbb{R}$
- $u \in \mathbb{R}^n$
- $\eta \sim \mathcal{N}\left(0, \gamma^2\right)$ for $\gamma \in \mathbb{R}$

A concrete realization with scalar u:

- $u = 2$
- $g = 3$
- $\gamma = 0.5$

- $y = 6.172$
- prior $\mathcal{N}(0, \Sigma_0 = 1)$

leads to a posterior with mean $\mu = \frac{(\Sigma_0 g)y}{\gamma^2 + \langle g, \Sigma_0 g \rangle} \approx 2$, which is what we see when we plot the result 3. The pCN-Sampler with $\beta = 0.25$ had an acceptance rate of 0.567.

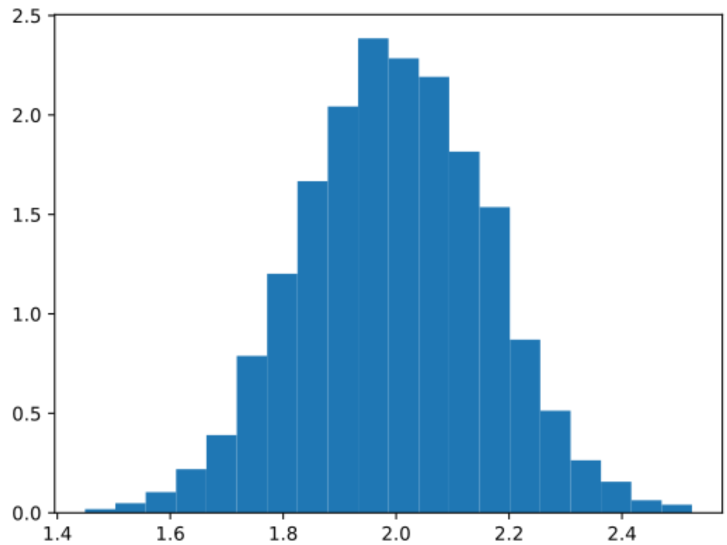

Figure 3: $N = 5000, \mu \approx 2$

For $n > 2$, the resulting posterior can not be plotted anymore. However, it is still Gaussian with given mean & covariance. Can just compare the analytical values to the sample values. Verify that the error decays like $\frac{1}{\sqrt{N}}$.

3.3 Bayesian inverse problem for $\mathcal{G}(u) = g(u + \beta u^3)$

Since the observation operator is not linear anymore, the resulting posterior is not Gaussian in general. However, since the dimension of the input u is 1, it can still be plotted.

The concrete realization with:

- $g = [3, 1]$

- $u = 0.5$
- $\beta = 0$
- $y = [1.672, 0.91]$
- $\gamma = 0.5$
- $\eta \sim \mathcal{N}\left(0, \gamma^2 I\right)$
- prior $\mathcal{N}\left(0, \Sigma_0 = 1\right)$

however leads to a Gaussian thanks to $\beta = 0$. The mean is $\mu = \frac{\langle g, y \rangle}{\gamma^2 + |g|^2} \approx 0.58$. Plot: 4

The pCN-Sampler with $\beta = 0.25$ (different beta) had an acceptance rate of 0.576.

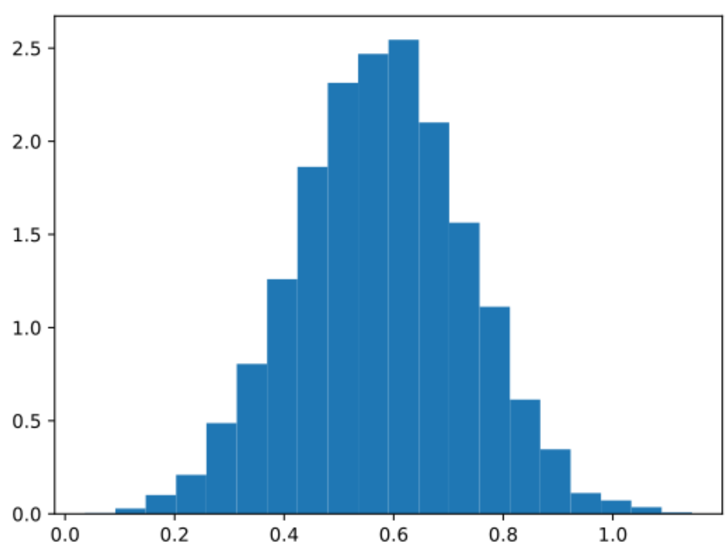

Figure 4: $N = 5000, \mu \approx 0.58$

For $\beta \neq 0$, the resulting posterior is not a Gaussian. Still $n = 1$, so it can be plotted. Just numerically normalize the analytical expression of the posterior?

3.4 Geophysics example: Lorenz-96 model

3.4.1 Based on:

Lorenz, E. N. (1996). Predictability—A problem partly solved. In Reprinted in T. N. Palmer & R. Hagedorn (Eds.), Proceedings Seminar on Predictability, Predictability of Weather and Climate, Cambridge UP (2006) (Vol. 1, pp. 1–18). Reading, Berkshire, UK: ECMWF.

3.4.2 Equation

A system of ODEs, representing the coupling between slow variables X and fast, subgrid variables Y.

$$\frac{\mathrm{d}X_k}{\mathrm{d}t} = -X_{k-1}(X_{k-2} - X_{k+1}) - X_k + F - hc\bar{Y}_k \tag{2}$$

$$\frac{1}{c}\frac{\mathrm{d}Y_{j,k}}{\mathrm{d}t} = -bY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - Y_{j,k} + \frac{h}{J}X_k \tag{3}$$

- $X = [X_0, ..., X_{K-1}] \in \mathbb{R}^K$
- $Y = [Y_{j,0}|...|Y_{j,K-1}] \in \mathbb{R}^{J\times K}$
  $Y_{j,k} = [Y_{0,k}, ..., Y_{J-1,k}] \in \mathbb{R}^J$
- $\bar{Y}_k = \frac{1}{J}\sum_j Y_{j,k}$
- periodic: $X_K = X_0$, $Y_{J,k} = Y_{0,k}$
- Parameters $\Theta = [F, h, c, b]$
- h: coupling strength
- c: relative damping
- F: external forcing of the slow variables (large scale forcing)
- b: scale of non-linear interaction of fast variables
- $t = 1 \Leftrightarrow 1$ day (simulation duration is given in days)

b or J? In the original paper, the equations are given in a different form, namely all explicit occurences of J above (in the fast-slow interaction) are replaced by b. Since in both concrete realizations (1996 & 2017) are identical and conviniently have $b = J = 10$, the difference doesn't lead to different results for that setup.

Where is F? In the original paper (at least the one I found), the forcing term F for the slow variables is missing in the equation, however it is referenced in the text, as well as present in a simpler (only slow variables) setup.

Maybe I have a shitty version of the paper?

"Looking ahead" vs. "Looking back" Comparing nonlinearity terms

$$-X_{k-1}(X_{k-2} - X_{k+1})$$

$$-bY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k})$$

for a given Y_k, does the "direction" of the $Z_{k\pm1}Z_{k\pm2}$ $(Z = X, Y)$ matter?

I don't think so, since the interaction with the other variable is only via point-value and average, and the nonlinearity is periodic.

A bit more formally: The PDE is invariant under "reversing" of the numbering: $Y_{j,k} \to Y_{J-j,k}$ which is the same as switching $+ \leftrightarrow -$ in the only "asymmetric" term.

Addendum 2 days later: Need to define more clearly what it means for the direction to <u>matter</u>. In the original paper on page 12, it is described how "active areas [..] propagate slowly eastward", while "convective activity tends to propagate westward within the active areas" (rephrased from paper). The paper also explicitly mentions the signs of the subscripts in that context. So some characteristics of the solution are definitely affected. What about the stuff we care about (statistical properties, chaotic behaviour)?

Initial conditions for the MCMC observation operator In [2], they describe their algorithm as:

"The objective function for each sample is accumulated over T = 100 days, using the end state of the previous forward integration as initial condition for the next one, without discarding any spin-up after a parameter update."

(Disregarding the fact that for this chaotic ODE and such a long simulation time the initial conditions are virtually irrelevant)

From a theoretical stand-point, should the observation operator $\mathcal{G}(u)$ not depend <u>only</u> on u, the parameters we try to estimate in the algorithm? And should it not be deterministic, all the (stochastic) observational noise encoded in the error term η?

Does this changing of the IC every iteration not pose a problem for the convergence of the algorithm?

Also does the spin-up they mention refer to discarding the first part of the simulation when computing steady-state averages? It somehow doesn't make sense to talk about a Markov-spin-up in the context of a single ODE-simulation, but ALSO it doens't make sense to talk about averages since from my understanding the MCMC-algorithm doesn't have to compute averages, it just looks at the final state?

3.4.3 Properties

- For $K = 36$, $J = 10$ and $\Theta = [F, h, c, b] = [10, 1, 10, 10]$ there is chaotic behaviour.
- The nonlinearities conserve the energies within a subsystem:
  - $E_X = \sum_k X_k^2$
    $\frac{1}{2}\frac{\mathrm{d}(\sum_k X_k^2)}{\mathrm{d}t} = \sum_k X_k \frac{\mathrm{d}X_k}{\mathrm{d}t} = -\sum_k (X_k X_{k-1} X_{k-2} - X_{k-1} X_k X_{k+1}) = 0$,
    where the last equality follows from telescoping + periodicity
  - $E_{Y_k} = \sum_j Y_{j,k}^2$
    which follows analogously to the X -case
- The interaction conserves the total energy:
  - $E_T = \sum_k (X_k^2 + \sum_j Y_{j,k}^2)$
    $\frac{1}{2}\frac{\mathrm{d}E_T}{\mathrm{d}t} = \sum_k X_k \frac{\mathrm{d}X_k}{\mathrm{d}t} + \sum_j Y_{j,k} \frac{\mathrm{d}Y_j,k}{\mathrm{d}t} = \sum_k X_k (-\frac{hc}{J} \sum_j Y_{j,k}) + \sum_j Y_{j,k} (\frac{hc}{J} X_k) = \sum_k -\frac{hc}{J} X_k (\sum_j Y_{j,k} + \frac{hc}{J} X_k (\sum_j Y_{j,k})) = 0$
- In the statistical steady state, the external forcing F (as long as its positive) balances the dampling of the linear terms.
- Averaged quantities
  - $$\langle \phi \rangle = \frac{1}{T} \int_{t_0}^{t_0+T} \phi(t)\, \mathrm{d}t$$
    (or a sum over discrete values)
  - Long-term time-mean in the statistical steady state: $\langle \cdot \rangle_\infty$
  - $$\left\langle X_k^2 \right\rangle_\infty = F \left\langle X_k \right\rangle_\infty - hc \left\langle X_k \bar{Y_k} \right\rangle_\infty \forall k \tag{4}$$
    (multiply X -equation by X, all X_k s are statistically equivalent, $\frac{\mathrm{d}\langle X \rangle}{\mathrm{d}t} = 0$ in steady state)

–

$$\left\langle \bar{Y_k^2} \right\rangle_\infty = \frac{h}{J} \left\langle X_k \bar{Y_k} \right\rangle_\infty \forall k \tag{5}$$

3.4.4 Model implementation

Implementing the model in python and using a locally 5-th order RK solver yields the following results (inital conditions are just uniformly random numbers in $[0, 1)$ since they don't matter for the long-term evolution of the chaotic system):

Reproducing the results of the original paper Running the setup with $K = 36, J = 10, (F, h, c, b) = (10, 1, 10, 10)$ given the following states 5, 6, which qualitatively agree with the results from Lorenz.

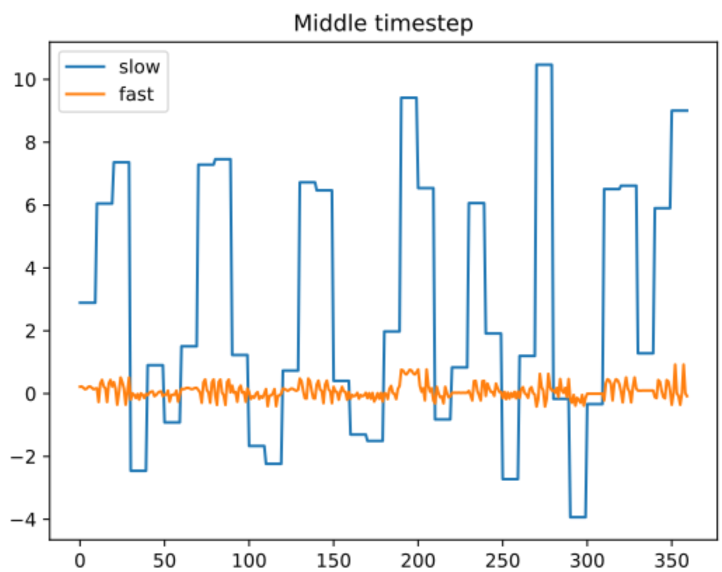

Figure 5: System after 30 days

The decay of the linear term and the forcing of the slow variables balance out after reaching the steady state, however there is a much bigger fluctuation in the energy than expected 7.

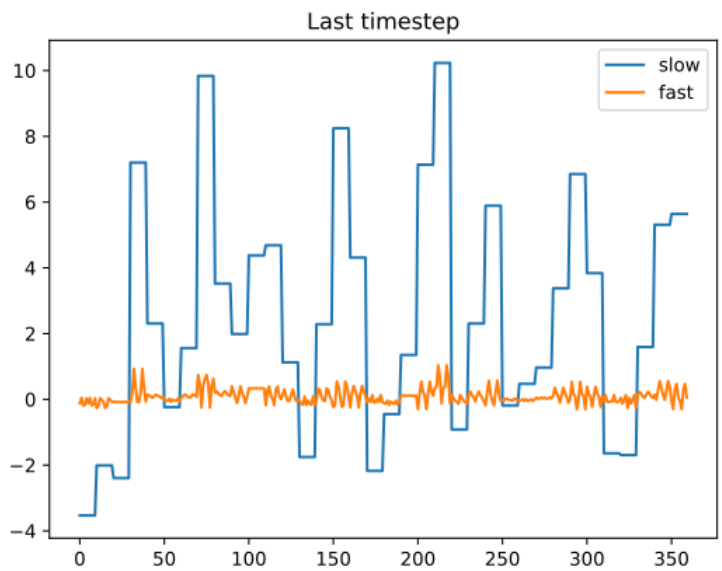

Figure 6: System after 60 days

Equilibrium averages Analysis suggests certain long-term averages to be equal in the equilibrium.

—————————- added Fri 22.05 ——————————

3.4.5 MCMC

Setup Denote the Lorenz-96 system 2, 3 with parameters $u = [F, h, c, b]$ as $\mathcal{M}[u]$. It acts on the initial condition $z_0 = [X_0, Y_0] \in \mathbb{R}^{K(J+1)}$ to evolve the system for N_t timesteps to generate the ? (find a good name for the "history" of the system) $Z = [{}^{X_1}_{Y_1}|\cdots|{}^{X_{N_t}}_{Y_{N_t}}] \in \mathbb{R}^{K(J+1)\times N_t}$:

$$Z = \mathcal{M}[u]z_0$$

The "moment function" $f(z) : \mathbb{R}^{K(J+1)} \to \mathbb{R}^{5K}$ (why can't we just use the state of the system directly?):

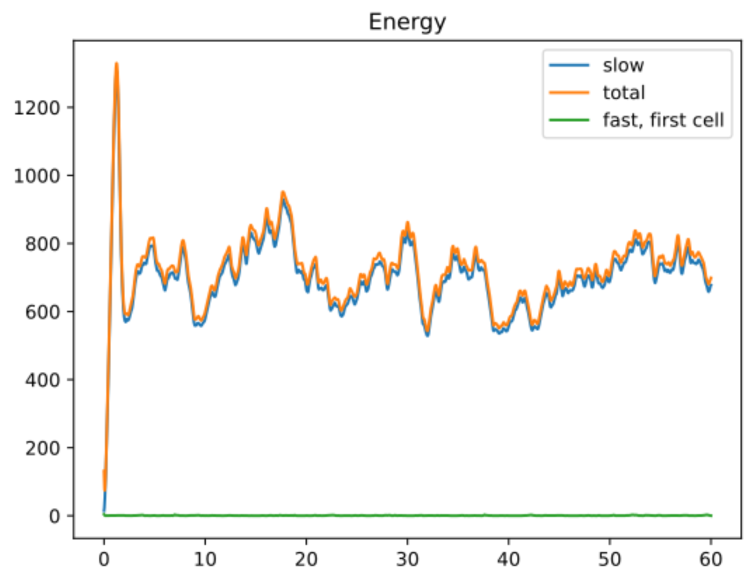

Figure 7: Energies in the system. $E_X >> E_{Y_k} > 0$

$$f(z) = \begin{bmatrix} X \\ \bar{Y} \\ X^2 \\ X\bar{Y} \\ \overline{Y^2} \end{bmatrix} \tag{6}$$

Defining a potential energy for a simulation with duration T

$$J(Z) = \frac{1}{2} \| \langle f(z) \rangle_T - \langle f(\tilde{z}) \rangle_\infty \|_\Sigma, \tag{7}$$

where $\tilde{z}$ is the output of a simulation with the true parameters $\tilde{u} = \Theta$ and $\langle \cdot \rangle_\infty$ is approximated by an average over time $\tilde{T} >> T$. Σ is the covariance matrix of the noise.

Putting it all together yields the following equation for the MCMC:

$$y = J(\mathcal{M}[u]z_0) + \eta$$

with:

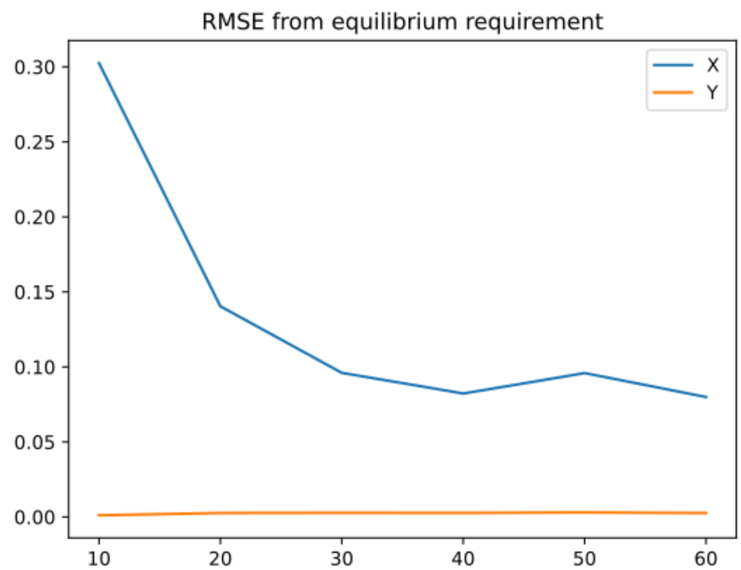

Figure 8: RMSE for long-term averages 4 and 5. Averaged over 10 runs

- $y \in \mathbb{R}^{5K}$: moment function of last state or mean of moment function?
- $u \in \mathbb{R}^4$: Parameter vector
- $J(\mathcal{M}[\cdot]z_0) : \mathbb{R}^4 \to \mathbb{R}^?$: Observational operator
- $\eta \sim \mathcal{N}(0, \Sigma)$, where $\Sigma = r^2[\mathrm{var}(f(\tilde{z}))]$

What is actually going on? They describe the noise with a covariance matrix $\Sigma \in \mathbb{R}^{5K \times 5K}$, implying that $\mathcal{G}(u) \in \mathbb{R}^{5K}$. However, in the description of the MCMC-algorithm they talk about computing the objective function, which I understand as meaning that $\mathcal{G}(u) \in \mathbb{R}$? Also the scariest thing is that python doesn't complain either way, there is some shady broadcasting going on in numpy.

Independently of the correctness of the simulation, there is a serious feasibility challenge As it stands now, doing a small run ($K = 6, J = 4, T = 10$) takes 14 seconds. This means a MCMC sampling takes approximately ∞.

The most time (1/3 of the total time) is spend calculating the nonlinear terms. They already use `np.roll` and no explicit loops, so I'm not confident that they can be sped up significantly. Maybe trying out something like `http://numba.pydata.org/` could give some speedup, I have no firsthand experience however.

Since the sampling is (approximately) embarassingly parallel, I think I could get a speedup of $\mathcal{O}(4)$ on my laptop.

References

[1] S. L. Cotter, G. O. Roberts, A. M. Stuart, and D. White. MCMC Methods for Functions: Modifying Old Algorithms to Make Them Faster. *Statistical Science*, 28(3):424–446, August 2013. Publisher: Institute of Mathematical Statistics.

[2] Tapio Schneider, Shiwei Lan, Andrew Stuart, and João Teixeira. Earth System Modeling 2.0: A Blueprint for Models That Learn From Observations and Targeted High-Resolution Simulations. *Geophysical Research Letters*, 44(24):12,396–12,417, 2017. _eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2017GL076101.

[3] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, May 2010.