Introduction to recursive machine learning

Applications

Juan Pablo Carbajal
juanpablo.carbajal@ost.ch

Eastern Switzerland University of Applied Sciences OST
Institute for Energy Technology IET
Scientific Computing and Engineering Group SCE

January 2023 - Rapperswil

Recap and Q&A

Kalman filter

$$\text{while no measurement: predict} \begin{cases} \hat{\boldsymbol{m}}_k = \mathbf{A}_k \boldsymbol{m}_{k-1} \\ \hat{\mathbf{P}}_k = \mathbf{A}_k \mathbf{P}_{k-1} \mathbf{A}_k^\top + \mathbf{Q}_{k-1} \end{cases}$$

$$\text{predict measurement} \begin{cases} \hat{\boldsymbol{y}}_k = \mathbf{H}_k \hat{\boldsymbol{m}}_k \\ \hat{\mathbf{S}}_k = \mathbf{H}_k \hat{\mathbf{P}}_k \mathbf{H}_k^\top + \mathbf{R}_k \end{cases}$$

$$\text{on measurement: update} \begin{cases} \mathbf{K}_k = \hat{\mathbf{P}}_k \mathbf{H}_k^\top \hat{\mathbf{S}}_k^{-1} \\ \boldsymbol{m}_k = \hat{\boldsymbol{m}}_k + \mathbf{K}_k \left(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\right) \\ \mathbf{P}_k = \hat{\mathbf{P}}_k - \mathbf{K}_k \hat{\mathbf{S}}_k \mathbf{K}_k^\top \end{cases}$$

Implement these functions in a programming language:

- `m, P` $\longleftarrow$ `kf_predict(m, P, A, Q)`
- `m, P, K` $\longleftarrow$ `kf_update(m, P, H, R, y)`
- `y, S` $\longleftarrow$ `kf_measure(m, P, H, R)`

1. Propagation of uncertainty to output using best estimate
2. Kalman gain
3. Update mean by projecting error on Kalman gain
4. Update cov by propagating uncertainty on measurement backward

Do not forget to update your measurement prediction after and update.

Which matrices participate in each function?

How would you determine which states are most sensitive to the residuals $\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k$?

Python & GNU Octave (MATLAB) tools

These libraries are relatively easy to follow, and have a book for documentation:

- FilterPy
  - `https://filterpy.readthedocs.io/en/latest/`
  - book: `https://nbviewer.org/github/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/table_of_contents.ipynb`
- ekfukf
  - MATLAB/Octave `https://github.com/EEA-sensors/ekfukf`,
  - Octave installable `https://github.com/kakila/ekfukf`
  - book `http://users.aalto.fi/~ssarkka/pub/cup_book_online_20131111.pdf` (if you like it, buy it!)

They are not performant (pykalman is abandoned), for prototyping.
Consider Julia: `https://github.com/JuliaGNSS/KalmanFilters.jl`

There are many libraries for Kalman filter, search for the one that suits you.
In my experience, I use simple ones for quickly prototyping, then implement what I need case specific. Algorithms are very simple.
Difficulty in inverting matrices: good algorithms are already available. Most relevant matrices are symmetric positive definite.

General probabilistic inference: PyMC3 `https://docs.pymc.io/en/stable/`

Warm-up

Use your code or a library to implement:

- recursive linear regression $y_k = t_k a + b + \epsilon_k$
- Delayed linear regression $y_k = w_0 + w_1 y_{k-1} + \cdots + w_d y_{k-d} + \epsilon_k$
- Extreme Learning Machines

$$y_k = \sum_{d=1}^{D} \Phi[\boldsymbol{y}_{k-d}]\,\boldsymbol{w}_d^\top$$

$$\Phi[\boldsymbol{x}] = \begin{bmatrix}\phi_1(\boldsymbol{x}) & \cdots & \phi_n(\boldsymbol{x})\end{bmatrix}, \quad \boldsymbol{w}_d = \begin{bmatrix}w_{d,1} & \cdots & w_{d,n}\end{bmatrix}$$

How does it differ from:

$$y_{d,k} = \Phi[y_{k-d}]\,\boldsymbol{w}^\top$$

Coffee break: 10 minutes

Participant examples

Examples provided by participants

- Frédéric Bless: Control of a heat pump test bench
- Rafael Graf: Incineration plant model validation. Inspired by `https://www.frontiersin.org/articles/10.3389/fenrg.2020.00049/full`
- Pirmin Weigele: Predictive damping in electromagnetic force compensation
- Matthias Frommelt: End-of-life of Li-ion batteries prediction using real-time data

Lunch: 1 hour

Discretization of ODEs

What's an ODE?

Ordinary differential equations (ODE) define a function (the unknown) via a relation of its derivatives:

$$\frac{\mathrm{d}^2 x(t)}{\mathrm{d}t^2} + \gamma \frac{\mathrm{d}x(t)}{\mathrm{d}t} + \omega^2 x(t) = w(t)$$

That's a **2nd oder** equation. For low order we can shorten it to:

$$\ddot{x} + \gamma \dot{x} + \omega^2 x = w(t)$$

High order equations can (almost always) be converted to a **system of 1st order** equations:

$$x_1(t) := x(t), \quad x_2(t) := \dot{x}(t)$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -\gamma \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t) = \mathbf{F}\boldsymbol{x} + \mathbf{L}w(t)$$

We exchange order for dimension.

Linear ODE

In general $\mathbf{F}$ and $\mathbf{L}$ can depend on $\boldsymbol{x}$ and time. If they are constant or depend only on time, the system is **linear** (affine):

$$
\begin{aligned}
\dot{\boldsymbol{x}} &= \mathbf{F}(t)\boldsymbol{x} + \mathbf{L}(t)w_x(t) \\
\dot{\boldsymbol{y}} &= \mathbf{F}(t)\boldsymbol{y} + \mathbf{L}(t)w_y(t) \\
\boldsymbol{z} &\coloneqq \boldsymbol{x} + a\boldsymbol{y} \\
\dot{\boldsymbol{z}} &= \dot{\boldsymbol{x}} + a\dot{\boldsymbol{y}} = \mathbf{F}(t)\boldsymbol{x} + \mathbf{L}(t)w_x(t) + a\mathbf{F}(t)\boldsymbol{y} + a\mathbf{L}(t)w_y(t) = \\
&= \mathbf{F}(t)\left(\boldsymbol{x} + a\boldsymbol{y}\right) + \mathbf{L}(t)\left(w_x(t) + aw_y(t)\right)
\end{aligned}
$$

The derivative of the combination follows the same equation with the combined inputs.

1st order homogenous ODE

Defines a 1-dimensional system: $\dot{x} = Fx$. Integrating both sides from 0 to t.

$$\int_0^t \dot{x}(\tau)\mathrm{d}\tau = x(t) - x(0) \rightarrow x(t) = x(0) + \int_0^t Fx(\tau)\mathrm{d}\tau$$

Recursively insert the formal solution in its expression:

$$x(t) = x(0) + \int_0^t F\left[x(0) + \int_0^\tau Fx(\tau_1)\mathrm{d}\tau_1\right]\mathrm{d}\tau = $$

$$= x(0) + Fx(0)t + \int_0^t \int_0^\tau F^2\left[x(0) + \int_0^{\tau_1} Fx(\tau_2)\mathrm{d}\tau_2\right]\mathrm{d}\tau_1\mathrm{d}\tau =$$

$$= x(0) + Fx(0)t + \frac{1}{2}F^2x(0)t^2 + \int_0^t \int_0^\tau \int_0^{\tau_1} F^3x(\tau_2)\mathrm{d}\tau_2\mathrm{d}\tau_1\mathrm{d}\tau = \dots$$

$$x(t) = x(0) + Fx(0)t + \frac{1}{2}F^2x(0)t^2 + \frac{1}{6}F^3x(0)t^3 + \dots =$$

$$= \left(1 + Ft + \frac{F^2t^2}{2!} + \frac{F^3t^3}{3!} + \dots\right)x(0) = \exp(Ft)x(0)$$

The infinite sum inside the parenthesis is the Taylor expansion of $\exp(Ft)$. Check it by expanding e^x around 0, i.e.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!}\frac{\mathrm{d}^n f}{\mathrm{d}x^n}\bigg|_{x_o}(x - x_o)^n$$

is the Taylor expansion of $f(x)$ around x_o

$$= x(0) + Fx(0)t + \int_0^t \int_0^\tau F^2x(\tau_1)\mathrm{d}\tau_1\mathrm{d}\tau =$$

1st order N-dimensional ODE

Homogenous

The homogenous case, defines a N-dimensional system: $\dot{\boldsymbol{x}} = \mathbf{F}\boldsymbol{x}$. Following the same procedure as before

$$\boldsymbol{x}(t) = \left(\mathbf{I} + \mathbf{F}t + \frac{\mathbf{F}^2 t^2}{2!} + \frac{\mathbf{F}^3 t^3}{3!} + \dots\right)\boldsymbol{x}(0) := \exp(\mathbf{F}t)\boldsymbol{x}(0)$$

That's the definition of the **matrix exponential**

The matrix exponential is not the exponential of each entry in the matrix (element-wise exponential). Consider

$$\begin{aligned}\mathbf{F} &= \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\\ \mathbf{F}^2 &= \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} = \mathbf{0}\\ \rightarrow \mathbf{F}^n &= \mathbf{0}\end{aligned}$$

Then

$$\exp(\mathbf{F}t) = \mathbf{I} + \mathbf{F}t = \begin{bmatrix}1 & t\\ 0 & 1\end{bmatrix} \neq \begin{bmatrix}e^0 & e^t\\ e^0 & e^0\end{bmatrix} = \begin{bmatrix}1 & e^t\\ 1 & 1\end{bmatrix}$$

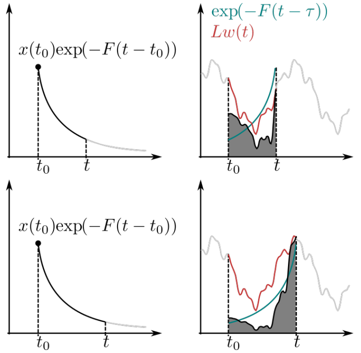
1st order N-dimensional ODE

Inhomogenous

For $\dot{\boldsymbol{x}} - \mathbf{F}\boldsymbol{x} = \mathbf{L}\boldsymbol{w}(t)$. We get the solution between t_0 and t:

$$\boldsymbol{x}(t) = \exp(\mathbf{F}(t - t_0))\boldsymbol{x}(0) + \int_{t_0}^{t} \exp(\boldsymbol{F}(t-\tau))\mathbf{L}\boldsymbol{w}(\tau)\mathrm{d}\tau$$

In the scalar case this looks like



Multiply both sides by $\exp(-\mathbf{F}t)$

$$\exp(-\mathbf{F}t)\dot{\boldsymbol{x}} - \exp(-\mathbf{F}t)\mathbf{F}\boldsymbol{x} = \exp(-\mathbf{F}t)\mathbf{L}\boldsymbol{w}(t)$$

You can prove from the definition of the matrix exponential the property

$$\frac{\mathrm{d}}{\mathrm{d}t}\exp(-\mathbf{F}t) = -\exp(-\mathbf{F}t)\mathbf{F}$$

Just like the scalar case, but here the order of multiplication is important. Observe that:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\exp(-\mathbf{F}t)\boldsymbol{x}(t)\right] = \exp(-\mathbf{F}t)\dot{\boldsymbol{x}} - \exp(-\mathbf{F}t)\mathbf{F}\boldsymbol{x} = \exp(-\mathbf{F}t)\left(\dot{\boldsymbol{x}} - \mathbf{F}\boldsymbol{x}\right)$$

Then the equation reduces to

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\exp(-\mathbf{F}t)\boldsymbol{x}(t)\right] = \exp(-\mathbf{F}t)\mathbf{L}(t)\boldsymbol{w}(t)$$

and by integrating both sides from t_0 to t we get the solution. Note that

$$\int_{t_0}^{t}\frac{\mathrm{d}}{\mathrm{d}\tau}\left[\exp(-\mathbf{F}\tau)\boldsymbol{x}(\tau)\right]\mathrm{d}\tau = \exp(-\mathbf{F}t)\boldsymbol{x}(t) - \exp(-\mathbf{F}t_0)\boldsymbol{x}(t_0)$$

Discretization

For the general ODE

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t)$$

If we move from t to $t + \Delta t$ we get the solution

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \int_t^{t+\Delta t} \boldsymbol{f}(\boldsymbol{x}(\tau), \tau)\mathrm{d}\tau$$

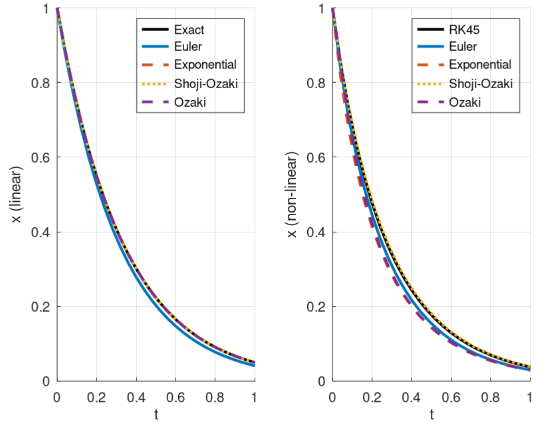
So the question is how to discretize the integral in the right-hand side. Choosing a quadrature defines an integration method. Euler method is obtained by the approximation

$$\int_t^{t+\Delta t} \boldsymbol{f}(\boldsymbol{x}(\tau), \tau)\mathrm{d}\tau \approx \boldsymbol{f}(\boldsymbol{x}(t), t)\Delta t$$

Discretization: example

Euler method 1D

$$\dot{x} = -3x \qquad \dot{x} = -3x - x^2$$



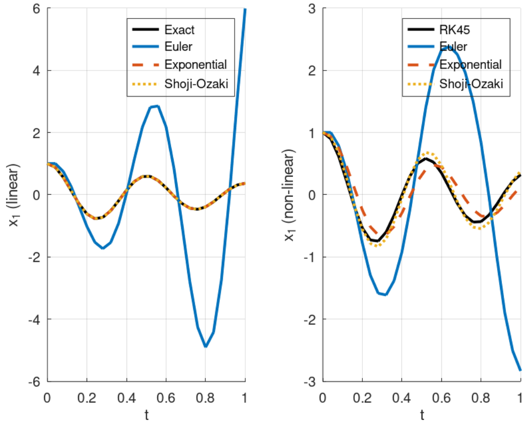
source: *s_euler_exp.m*

A comparison of discretization methods. We will see some of the others later on. Clearly Euler is not the best.

Discretization: example

Euler method 2D

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -\gamma \end{bmatrix} \boldsymbol{x} \qquad \dot{\boldsymbol{x}} = \begin{bmatrix} x_2 \\ -\omega^2 \sin(x_1) - \gamma x_2 \end{bmatrix}$$



source: *s_euler_exp.m*

Discretization
Linear time invariant (LTI) systems

The system $\dot{\boldsymbol{x}} - \mathbf{F}\boldsymbol{x} = \mathbf{L}\boldsymbol{w}(t)$ has an exact solution between t and $t + \Delta t$

$$\boldsymbol{x}(t) = \exp(\mathbf{F}\Delta t)\boldsymbol{x}(0) + \exp(\mathbf{F}\Delta t)\int_{t}^{t+\Delta t} \exp(\boldsymbol{F}(t-\tau))\mathbf{L}\boldsymbol{w}(\tau)\mathrm{d}\tau$$

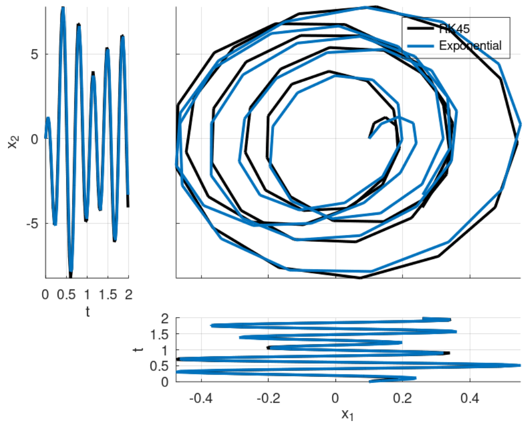
So the approximation is only for the last integral. For example:

$$\hat{\boldsymbol{x}}(t_{k+1}) = \exp(\mathbf{F}\Delta t)\left[\hat{\boldsymbol{x}}(t_k) + \mathbf{L}\boldsymbol{w}(t_k)\Delta t\right]$$

When $\boldsymbol{w}(t)$ is stochastic this is an instance of the Euler-Murayama method.

Discretization

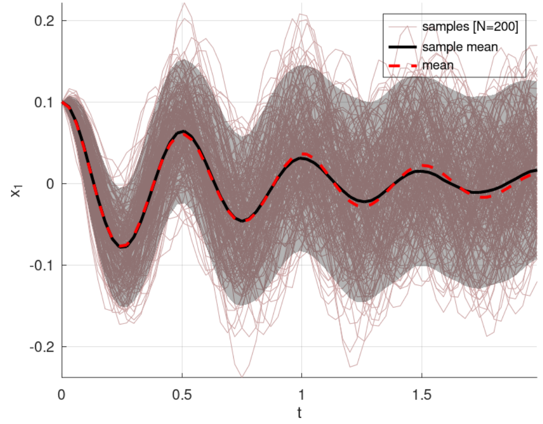
Linear time invariant (LTI) systems



source: *s_exp_step_input.m*

Discretization
Linear time invariant (LTI) systems

This also works when $\boldsymbol{w}(t)$ is a random process.



source: *s_exp_step_input.m*

When $\boldsymbol{w}(t)$ is stochastic this is an instance of the Euler-Murayama method. The stochastic case is subtle, e.g. the discretized noise has a variance that depends on the time step Δt. For details, refer to Simo Särkkä and Arno Solin (2019). Applied Stochastic Differential Equations.

There are stochastic methods that generalize the deterministic ones, e.g. Stochastic Runge-Kutta. For second order system check the Stochastic Verlet algorithm.

Polynomial regression

Hands-on

The data model is $y_k = p(t_k, n) + \epsilon_k$, where

$$p(t,n) := \sum_{s=0}^{n} a_s t^s$$

is a polynomial of degree n. Consider the system

$$\dot{x}_1 = x_2 \quad \dot{x}_2 = x_3 \quad \dot{x}_3 = 0$$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$$

if we start at $\boldsymbol{x}(0) = \begin{bmatrix}a_0 & a_1 & 2a_2\end{bmatrix}$ we get

$$x_3(t) = 2a_2 \quad x_2(t) = a_1 + 2a_2 t \quad x_1(t) = a_0 + a_1 t + a_2 t^2$$

we recover $p(t,2)$ on $x_1(t)$!

Polynomial regression
Hands-on

Can be generalized to any finite degree:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{F}\boldsymbol{x}, \qquad F_{ij} = \delta_{i(j+1)},\ 1 \leq i, j \leq n+1 \\ \boldsymbol{x}(0)^\top &= \begin{bmatrix} a_0 & a_1 & 2a_2 & \dots & n!a_n \end{bmatrix} \end{aligned}$$

has $p(t, n)$ in $x_1(t)$.
Then filter

$$\begin{aligned} \mathbf{A} &= \exp\left(\mathbf{F}\Delta t\right), \qquad \mathbf{Q} \neq \mathbf{0} \\ \mathbf{H} &= \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}, \qquad \mathbf{R} \neq \mathbf{0} \end{aligned}$$

implements polynomial regression with drift model in the coefficients.
See `s_polyreg.py`

Coffee break: 10 minutes

Parameter estimation

What to estimate?

The linear Kalman filter is defined with the parameters $\mathbf{A}, \mathbf{H}, \mathbf{Q}, \mathbf{R}, \mathbf{P}_0, \boldsymbol{m}_0$.

- $\mathbf{A}, \mathbf{H}$ reflect knowledge of the physical system (unless data-driven).
- Bounds for the diagonal of $\mathbf{R}$ can be estimated from the data.
- $\mathbf{Q}$ in general is hidden, unless derived from physical SDE.
- $\mathbf{P}_0, \boldsymbol{m}_0$ can be critical for the performance of filters with long-term memories (eigenvalues of $\mathbf{A}$ closer to or bigger than 1).

Idea: start by freezing degrees of freedom (à la stochastic gradient decent).

To estimate $\mathbf{R}$ use any smoother on the data and compute the variance of the residuals between smoothed data and raw data. There is almost always a strong expectation on this parameter.

Many physics based models will have limited memory. An exception would be a fluid model where waves can carry information for long distances and induce long-term temporal correlations. In general systems with echo (transport, waves) can produce these effects.

Posterior likelihood

Bayesian filter for state space models with parameter vector $\boldsymbol{\theta}$

$$\begin{aligned}
&\boldsymbol{\theta} \sim p(\boldsymbol{\theta}), \quad \boldsymbol{x}_o \sim p(\boldsymbol{x}_0|\boldsymbol{\theta}) \\
&\boldsymbol{x}_k \sim p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{\theta}) \\
&\boldsymbol{y}_k \sim p(\boldsymbol{y}_k|\boldsymbol{x}_k, \boldsymbol{\theta})
\end{aligned}$$

The idea is to compute the posterior distribution of states and parameters given the measurements

$$p(\boldsymbol{x}_{0:T}, \boldsymbol{\theta}|\boldsymbol{y}_{1:T}) \propto p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{0:T}, \boldsymbol{\theta}) p(\boldsymbol{x}_{0:T}|\boldsymbol{\theta}) p(\boldsymbol{\theta})$$

where

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{0:T}, \boldsymbol{\theta}) = \prod_{k=1}^{T} p(\boldsymbol{y}_k|\boldsymbol{x}_k, \boldsymbol{\theta})$$

$$p(\boldsymbol{x}_{0:T}|\boldsymbol{\theta}) = p(\boldsymbol{x}_0|\boldsymbol{\theta}) \prod_{k=1}^{T} p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{\theta})$$

We focus on the parameters, so we "average" (marginalize) out the states.

Herein follow chapter 12 of the book Särkkä, S. (2013). Bayesian Filtering and Smoothing.

These are the same relations as before, we just made the parameters explicit.

$p(\boldsymbol{\theta})$ is the prior over the parameters.

Posterior likelihood

Bayesian filter for state space models with parameter vector $\boldsymbol{\theta}$

$$\begin{aligned}
&\boldsymbol{\theta} \sim p(\boldsymbol{\theta}), \quad \boldsymbol{x}_o \sim p(\boldsymbol{x}_0|\boldsymbol{\theta}) \\
&\boldsymbol{x}_k \sim p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{\theta}) \\
&\boldsymbol{y}_k \sim p(\boldsymbol{y}_k|\boldsymbol{x}_k, \boldsymbol{\theta})
\end{aligned}$$

The posterior distribution of parameters given the measurements

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T}) \propto p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

where (check the "averaged" states)

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta}) = \prod_{k=1}^{T} p(\boldsymbol{y}_k|\boldsymbol{y}_{1:k-1}, \boldsymbol{\theta})$$

$$p(\boldsymbol{y}_k|\boldsymbol{y}_{1:k-1}, \boldsymbol{\theta}) = \int \overbrace{\underbrace{p(\boldsymbol{y}_k|\boldsymbol{x}_k, \boldsymbol{\theta})}_{\text{measurement}}\underbrace{p(\boldsymbol{x}_k|\boldsymbol{y}_{1:k-1}, \boldsymbol{\theta})}_{\text{prediction}}}^{\text{update}} \mathrm{d}\boldsymbol{x}_k$$

Herein follow chapter 12 of the book Särkkä, S. (2013). Bayesian Filtering and Smoothing.

For conveninece of the notation we used

$$p(\boldsymbol{y}_1|\boldsymbol{y}_{1:0}, \boldsymbol{\theta}) = p(\boldsymbol{y}_1|\boldsymbol{\theta})$$

To get the recursion for the posterior of the parameters, the key is to write a recursion for likelihood of the data. We do that next.

Posterior likelihood

We know how to compute the posterior distribution of parameters given the measurements at a given step k

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{1:k}) \propto p(\boldsymbol{y}_{1:k}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

We define log-likelihood $\phi_k(\boldsymbol{\theta})$:

$$\begin{aligned}\exp(\phi_k(\boldsymbol{\theta})) \coloneqq p(\boldsymbol{y}_{1:k}|\boldsymbol{\theta}) &= \prod_{n=1}^{k} p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1}, \boldsymbol{\theta}) \\ &= p(\boldsymbol{y}_k|\boldsymbol{y}_{k-1}, \boldsymbol{\theta}) \overbrace{\prod_{n=1}^{k-1} p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1}, \boldsymbol{\theta})}^{p(\boldsymbol{y}_{1:k-1}|\boldsymbol{\theta})=\exp(\phi_{k-1}(\boldsymbol{\theta}))} \\ &= p(\boldsymbol{y}_k|\boldsymbol{y}_{k-1}, \boldsymbol{\theta}) \exp(\phi_{k-1}(\boldsymbol{\theta}))\end{aligned}$$

Taking logs

$$\phi_k(\boldsymbol{\theta}) = \phi_{k-1}(\boldsymbol{\theta}) + \log\left[p(\boldsymbol{y}_k|\boldsymbol{y}_{k-1}, \boldsymbol{\theta})\right], \quad \phi_0(\boldsymbol{\theta}) = \log\left[p(\boldsymbol{\theta})\right]$$

Herein follow chapter 12 of the book Särkkä, S. (2013). Bayesian Filtering and Smoothing.

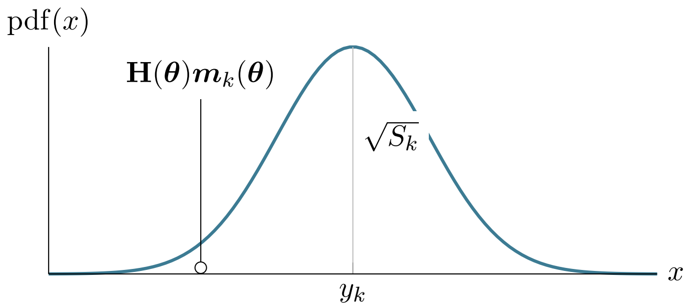
If the prior $p(\boldsymbol{\theta})$ is uniform, this is the same as maximum likelihood (ML). If the distribution reflect our prior knowledge about the parameters, then it is maximum a posteriori probability (MAP).
$\phi_k(\boldsymbol{\theta})$ is the log-likelihood, sometimes the negative is defined.
Maximizing the log-likelihood gives optimal estimates. This should remind you of Viterbi, if you know it.

Linear KF likelihood

With the filters we have been studying, the log-likelihood has an explicit formula.

$$\boldsymbol{v}_k(\boldsymbol{\theta}) \coloneqq \boldsymbol{y}_k - \mathbf{H}(\boldsymbol{\theta})\boldsymbol{m}_k(\boldsymbol{\theta})$$

$$\phi_k(\boldsymbol{\theta}) = \phi_{k-1}(\boldsymbol{\theta}) - \frac{1}{2}\log\left[2\pi \det \mathbf{S}_k(\boldsymbol{\theta})\right] - \frac{1}{2}\boldsymbol{v}_k^\top(\boldsymbol{\theta})\mathbf{S}_k^{-1}(\boldsymbol{\theta})\boldsymbol{v}_k(\boldsymbol{\theta})$$


$\mathrm{pdf}(x)$
$\mathbf{H}(\boldsymbol{\theta})\boldsymbol{m}_k(\boldsymbol{\theta})$
$\sqrt{S_k}$
x
y_k

To compute the likelihood we need to run over the whole data for each new value of $\boldsymbol{\theta}$. Each run over the whole data is done at fixed parameter value.

Maximum likelihood

Example: oscillator

$$\boldsymbol{x}_k = \exp\left(\begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \Delta t\right)\boldsymbol{x}_{k-1} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_{k-1}, \quad w_{k-1} \sim \mathcal{N}(0, q)$$

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \boldsymbol{x}_k + r_k, \quad r_k \sim \mathcal{N}(0, \sigma_y^2)$$

`s_estimate_oscillator.py` does

- Line search for ω
- Optimization for ω, q, σ_y, $\boldsymbol{m}_0$

Gradient based methods will not be sensitive to the data if started far away from a local maximum: likelihood tends to be sharp and wiggly, vanishing gradients. Alternative: informative priors, e.g. tight bounds on the parameters.

If you want to optimize many parameters of the model, check the Expectation Maximization algorithm.

Wrap-up

Summary and Outlook

- Linear Kalman filter with Gaussian distributions ✓
- Non-linear
  - Local linearization (Extended Kalman filter)
  - Gaussian approximation (Unscented Kalman filter, General Gaussian filters: GHKF, CKF)
  - Particle filters, . . .
  - Brute force (naive MC, simulation of trajectories)
- Other noise models (check generalizations of KF, Särkkä's book references)
- Recursive or Batch?
  - Data: fixed size or incoming?
  - Estimation parameters: recorded data (batch) or online?