LINEAR TIME GAUSSIAN PROCESSES

a.k.a. something we kind of always thought was true, but didn’t actually know... and other cool findings

John P. Cunningham
Columbia University
16 October 2020

Reading: General linear-time inference for Gaussian Processes on one dimension (Loper et al., 2020)
gaussian processes and state-space models

latent exponentially generated gaussian processes (leggp)

toolkit: cyclic reduction for parallelizing message passing in tridiagonal (gauss markov) models
gaussian processes and state-space models

latent exponentially generated gaussian processes (leggp)

toolkit: cyclic reduction for parallelizing message passing in tridiagonal (gauss markov) models
The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:
  1. write GMP in Langevin/SDE form:
     \[
     z(t) = z(0) + \int_0^t (-Gz(s) \, ds + \sigma dw(s))
     \]
     or...
     \[
     \begin{bmatrix}
     df(t) \\
     \vdots \\
     df_{m-1}(t)
     \end{bmatrix}
     = -G
     \begin{bmatrix}
     f(t) \\
     \vdots \\
     f_{m-1}(t)
     \end{bmatrix}
     + \sigma dw(t)
     \]
  2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.
  3. Kalman filtering (aka RTS) will do inference in linear time: message passing, tridiagonalization, etc.
  4. a variety of papers play this game; eg Gilboa, Saatci, Cunningham (2013) ICML, (2015) IEEE PAMI.

- So how good is this game (for stationary 1d GP)?
How complex are Gaussian processes?

The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
How complex are Gaussian processes?

The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:
The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).

- lots of us have had good success with gauss-markov processes as approximations. Reminder:

   1. write GMP in Langevin/SDE form:

   $z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s))$ or...

   $\begin{bmatrix}
   \frac{df(t)}{dt} \\
   \vdots \\
   \frac{df^{m-1}(t)}{dt^{m-1}} \\
   \end{bmatrix}
   = -G
   \begin{bmatrix}
   \frac{f(t)}{dt} \\
   \vdots \\
   \frac{f^{m-1}(t)}{dt^{m-1}} \\
   \end{bmatrix}
   + \sigma dw(t)$

   2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.

   3. Kalman filtering (aka RTS) will do inference in linear time: message passing, tridiagonalization, etc.

   4. a variety of papers play this game; eg Gilboa, Saatci, Cunningham (2013) ICML, (2015) IEEE PAMI.
The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:

1. write GMP in Langevin/SDE form:

$$z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \quad \text{or...} \quad \begin{bmatrix} \frac{df(t)}{dt} \\
\vdots \\
\frac{df^{m-1}(t)}{dt^{m-1}} \\
\frac{df^{m}(t)}{dt^{m}} \\
\frac{dt^{m}}{dt^{m-1}} \\
\frac{dt^{m-1}}{dt^{m-2}} \\
\vdots \\
\frac{dt^{1}}{dt^{0}} \end{bmatrix} = -G \begin{bmatrix} \frac{f(t)}{dt} \\
\vdots \\
\frac{df^{m-1}(t)}{dt^{m-1}} \\
\frac{df^{m}(t)}{dt^{m}} \\
\frac{dt^{m}}{dt^{m-1}} \\
\frac{dt^{m-1}}{dt^{m-2}} \\
\vdots \\
\frac{dt^{1}}{dt^{0}} \end{bmatrix} + \sigma dw(t)$$

2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.
The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:
  
  1. write GMP in Langevin/SDE form:

\[
\begin{align*}
  z(t) &= z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \\
  &= -G \begin{bmatrix}
  \frac{df(t)}{dt} \\
  \vdots \\
  \frac{df^{m-1}(t)}{dt^{m-1}}
\end{bmatrix}
  + \sigma dw(t)
\end{align*}
\]

  or...

\[
\begin{bmatrix}
  \frac{df(t)}{dt} \\
  \vdots \\
  \frac{df^{m-1}(t)}{dt^{m-1}}
\end{bmatrix}
  = -G \begin{bmatrix}
  \frac{f(t)}{dt} \\
  \vdots \\
  \frac{df^{m-1}(t)}{dt^{m-1}}
\end{bmatrix}
\]

2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.

3. Kalman filtering (aka RTS) will do inference in linear time: message passing, tridiagonalization, etc.
How complex are Gaussian processes?

The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:

  1. write GMP in Langevin/SDE form:

      $$z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s))$$

      or...

      $$\begin{bmatrix}
      \frac{df(t)}{dt} \\
      \vdots \\
      \frac{df_{m-1}(t)}{dt_{m-1}} \\
      \frac{dt_{m-1}}{dt_{m}}
      \end{bmatrix}
      = -G
      \begin{bmatrix}
      \frac{f(t)}{dt} \\
      \vdots \\
      \frac{df_{m-1}(t)}{dt_{m-1}} \\
      \frac{dt_{m-1}}{dt_{m}}
      \end{bmatrix}
      + \sigma dw(t)$$

  2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.

  3. Kalman filtering (aka RTS) will do inference in linear time: message passing, tridiagonalization, etc.

  4. a variety of papers play this game; eg Gilboa, Saatci, Cunningham (2013) ICML, (2015) IEEE PAMI.
The usual refrain: GP have $O(n^2)$ storage and $O(n^3)$ runtime complexity...and yet:

- ARMA(4,4) models work just great (remark: we should do a more honest job wrt statsmodels, etc.).
- lots of us have had good success with gauss-markov processes as approximations. Reminder:
  1. write GMP in Langevin/SDE form:

\[
    z(t) = z(0) + \int_{0}^{t} (-Gz(s)ds + \sigma dw(s))
\]

\[
    \begin{bmatrix}
        \frac{df(t)}{dt} \\
        \vdots \\
        \frac{df_{m-1}(t)}{dt} \\
        \frac{df_{m}(t)}{dt} \\
    \end{bmatrix}
    =
    -G
    \begin{bmatrix}
        \frac{f(t)}{dt} \\
        \vdots \\
        \frac{f_{m-1}(t)}{dt} \\
        \frac{f_{m}(t)}{dt} \\
    \end{bmatrix}
    + \sigma dw(t)
\]

2. Matern kernels have this representation (exact); SE, RQ, etc have good approximate representations.
3. Kalman filtering (aka RTS) will do inference in linear time: message passing, tridiagonalization, etc.
4. a variety of papers play this game; eg Gilboa, Saatci, Cunningham (2013) ICML, (2015) IEEE PAMI.

- So how good is this game (for stationary 1d GP)?
Let’s be a bit careful

- The OU formulation from the last slide is clean...

\[ z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \]

- ... but it’s a hassle: stability $\leftrightarrow R(eig(G)) > 0$, which is a nontrivial constraint, and tough to consider theoretically.
Let’s be a bit careful

- The OU formulation from the last slide is clean...

\[ z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \]

- ... but it’s a hassle: stability \( \leftrightarrow R(eig(G)) > 0 \), which is a nontrivial constraint, and tough to consider theoretically.

- So we have some work to do.
Let’s be a bit careful

- The OU formulation from the last slide is clean...
  \[ z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \]

- ... but it’s a hassle: stability \( \leftrightarrow \mathcal{R}(\text{eig}(G)) > 0 \), which is a nontrivial constraint, and tough to consider theoretically.

- So we have some work to do.

- Define the following state-space model:
  \[
  z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^T + R - R^T \right) z(s)ds + Ndw(s) \right)
  \\
x(t) | z(t) \sim \mathcal{N} \left( Bz(t) , \Lambda\Lambda^T \right).
  \]
Let’s be a bit careful

- The OU formulation from the last slide is clean...

\[ z(t) = z(0) + \int_0^t (-Gz(s)ds + \sigma dw(s)) \]

- ... but it’s a hassle: stability $\leftrightarrow R(eig(G)) > 0$, which is a nontrivial constraint, and tough to consider theoretically.

- So we have some work to do.

- Define the following state-space model:

\[
\begin{align*}
  z(t) &= z(0) + \int_0^t \left(-\frac{1}{2} \left(NN^\top + R - R^\top\right)z(s)ds + Ndw(s)\right) \\
  x(t)|z(t) &\sim \mathcal{N}\left(Bz(t), \Lambda\Lambda^\top\right).
\end{align*}
\]

- We will call this SSM a latent exponentially generated GP, $x \sim \text{LEG}(N, R, B, \Lambda)$. We’ll get back to this...
HOW COMPLEX ARE GAUSSIAN PROCESSES?
How complex are Gaussian processes?

- iid gaussians
- stationary gp
- state space models

How much gap is here?
How complex are Gaussian processes?

- IID Gaussians
- Stationary GP
How complex are Gaussian processes?

- iid gaussians
- stationary gp
- SSM of rank 1

Cunningham Lab, Columbia University
How complex are Gaussian processes?

- iid gaussians
- Stationary gp
- SSM of rank 2
- SSM of rank 1
- iid gaussians
How complex are Gaussian processes?

- iid gaussians
- Stationary GP
- SSM of rank 2
- SSM of rank 1
- SSM of rank $\ell$
- ...
How complex are Gaussian processes?

- iid gaussians
- stationary gp
- SSM of rank 2
- SSM of rank 1
- SSM of rank $\ell$

... }

None!
That’s it

Any stationary, univariate GP is a *linear-time* object. More specifically:

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

Banter:

▶ we (the GP community) did not know that already... but yeah probably we thought it was true.
▶ spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
▶ astrostat has *celerite* kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.
▶ just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.
▶ idea: let’s do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank \( \ell \).

**Intuition:**
1. LEG is a mixture of \( \ell \) general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

**Banter:**
- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, \( \gg \) an appeal to Bochner).
- astrostat has celerite kernels, which are \( \approx \) block-diagonal LEGGP, without theoretical results.
- just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.
- idea: let’s do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- **Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank \( \ell \).**

Intuition:

- LEG is a mixture of \( \ell \) general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
- All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
- Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
- Thus, LEG are general and linear time.

Banter:

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, \( \gg \) an appeal to Bochner).
- astrostat has [celerite](#) kernels, which are \( \approx \) block-diagonal LEGGP, without theoretical results.
- just in case we don't get to it: cyclic reduction to parallelize SSM is really cool.
- idea: let's do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

**Intuition:**

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.

**Banter:**

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
- astrostat has *celerite* kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.
- just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.
- idea: let’s do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
That’s it

Any stationary, univariate GP is a linear-time object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

Intuition:

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.

2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
Any stationary, univariate GP is a *linear-time* object. More specifically:

▶ Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

Intuition:

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
Any stationary, univariate GP is a *linear-time* object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

Intuition:

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.
Any stationary, univariate GP is a \textit{linear-time} object. More specifically:

- \textit{Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank }$\ell$.\textit{ }

\textbf{Intuition:}

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.

2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).

3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).

4. Thus, LEG are general and linear time.

\textbf{Banter:}
Any stationary, univariate GP is a linear-time object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

Intuition:

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

Banter:

- we (the GP community) did not know that already... but yeah probably we thought it was true.

spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ appeal to Bochner).

astrostat has celerite kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.

just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.

idea: let’s do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.

**Intuition:**

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

**Banter:**

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
Any stationary, univariate GP is a \textit{linear-time} object. More specifically:

- \textit{Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.}

**Intuition:**

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

**Banter:**

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
- astrostat has \textit{celerite} kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- *Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank $\ell$.*

**Intuition:**

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.

2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).

3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).

4. Thus, LEG are general and linear time.

**Banter:**

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
- astrostat has *celerite* kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.
- just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.
Any stationary, univariate GP is a *linear-time* object. More specifically:

- *Any stationary continuous kernel can be approximated to arbitrary accuracy with a LEGGP of certain rank* $\ell$.

**Intuition:**

1. LEG is a mixture of $\ell$ general, well-behaved (stable, stationary, correlated) linear-gaussian state-space components.
2. All vector-valued spectral mixture kernels are LEG kernels (who knew?! spectral mixtures are SSM).
3. Vector-valued spectral mixture kernels are general (total variation convergence; see Thm 1).
4. Thus, LEG are general and linear time.

**Banter:**

- we (the GP community) did not know that already... but yeah probably we thought it was true.
- spectral mixture kernels are extended to complex-matrix-valued spectra (nontrivial, $\gg$ an appeal to Bochner).
- astrostat has *celerite* kernels, which are $\approx$ block-diagonal LEGGP, without theoretical results.
- just in case we don’t get to it: cyclic reduction to parallelize SSM is really cool.
- idea: let’s do a deep dive on cyclic reduction, pivoted cholesky, and multigrid methods.
gaussian processes and state-space models

latent exponentially generated gaussian processes (leegp)

toolkit: cyclic reduction for parallelizing message passing in tridiagonal (gauss markov) models
Understanding PEG via samples

\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} (NN^\top + R - R^\top) z(s) ds + Ndw(s) \right) \]

- rank \( \ell = 2 \) (first dim plotted)
- \( NN^\top \) is a diffusion; increasing any eigenvalue makes that direction less predictable.
- \( R - R^\top \) is a rotation: eigenvalues correspond to frequency, vectors to mixing into \( z \).
The process:

\[
\begin{align*}
    z(t) &= z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s) ds + Ndw(s) \right) \\
    x(t) | z(t) &\sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^\top \right)
\end{align*}
\]
FROM PEG to LEG

The process:

\[
z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^T + R - R^T \right) z(s) ds + Ndw(s) \right)
\]

\[
x(t)|z(t) \sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^T \right)
\]

Its well-behaved covariance:

\[
C_{PEG}(\tau; N, R) \triangleq \exp \left( -\frac{\tau}{2} \left( NN^T + R - R^T \right) \right)
\]

\[
C_{LEG}(\tau; N, R, B, \Lambda) \triangleq B \left( C_{PEG}(\tau; N, R) \right) B^T + \delta_{\tau=0} \Lambda \Lambda^T
\]
FROM PEG TO LEG

The process:

\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^T + R - R^T \right) z(s) ds + Ndw(s) \right) \]

\[ x(t) | z(t) \sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^T \right) \]

Its well-behaved covariance:

\[ C_{PEG}(\tau; N, R) \triangleq \exp \left( -\frac{\tau}{2} \left( NN^T + R - R^T \right) \right) \]

\[ C_{LEG}(\tau; N, R, B, \Lambda) \triangleq B \left( C_{PEG}(\tau; N, R) \right) B^T + \delta_{\tau=0} \Lambda \Lambda^T \]

Notes:

- LEG is just a state-space model
- Linear mixture of state-space components (very standard)
- These derivations are nontrivial and it’s worth reading the appendix to see how deep this work goes.
- LEG is closed under addition (non obvious but good to know)
is LEG linear?

\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s) ds + Ndw(s) \right) \]

\[ x(t) | z(t) \sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^\top \right) \]
is LEG linear?

\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} (NN^\top + R - R^\top) z(s) ds + Ndw(s) \right) \]

\[ x(t) | z(t) \sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^\top \right) \]

▶ plenty of extensions possible; see §4 of the paper.
▶ linear run-time, as promised. Also, \( 10^9 \) is a big number.
**IS LEG ACCURATE?**

\[
z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s) ds + Ndw(s) \right)
\]

\[
x(t) \mid z(t) \sim \mathcal{N} \left( Bz(t), \Lambda\Lambda^\top \right)
\]
is LEG accurate?

\[
z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s) ds + Ndw(s) \right)
\]

\[
x(t)|z(t) \sim \mathcal{N} \left( Bz(t), \Lambda\Lambda^\top \right)
\]

\[\ell = 4\] is not bad!

\[\text{well, we already knew Matern } \nu = \frac{1}{2} \text{ is a SSM; in fact it's a rank-2 LEG.}\]

\[\text{RQ also quite well considering how different it is from the LEG in terms of spectra...}\]
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{LEG}(\tau) = B \exp \left( -\frac{\tau}{2} \left( NN^T + R - R^T \right) \right) B^T + \delta_{\tau=0} \Lambda \Lambda^T \]
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{\text{RQ}}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{\text{LEG}}(\tau) = B \exp \left( -\frac{\tau}{2} \left( N N^\top + R - R^\top \right) \right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad vs \quad C_{LEG}(\tau) = B \exp\left( -\frac{\tau}{2} \left( NN^\top + R - R^\top \right) \right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{LEG}(\tau) = B \exp \left( -\frac{\tau}{2} \left( NN^\top + R - R^\top \right) \right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- ...but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{\text{RQ}}(\tau) = \frac{2}{(1 + \tau^2)} \quad \text{vs} \quad C_{\text{LEG}}(\tau) = B \exp\left(-\frac{\tau}{2} \left( NN^\top + R - R^\top \right)\right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- …but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel

This apparent contradiction is resolved by considering the different timescales involved in any Gaussian Process:
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{\text{RQ}}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{\text{LEG}}(\tau) = B \exp \left( -\frac{\tau}{2} \left( NN^\top + R - R^\top \right) \right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- ...but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel

This apparent contradiction is resolved by considering the different timescales involved in any Gaussian Process:

- LEG uniformly approximate any stationary covariance \(\rightarrow\) uniformly accurate predictions at \textit{any fixed timescale}
RQ excerpts that I found useful for discussion

How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{LEG}(\tau) = B \exp\left(-\frac{\tau}{2} (NN^\top + R - R^\top)\right) B^\top + \delta_{\tau=0}\Lambda\Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- ...but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel

This apparent contradiction is resolved by considering the different timescales involved in any Gaussian Process:

- LEG uniformly approximate any stationary covariance \( \rightarrow \) uniformly accurate predictions at any fixed timescale
- LEG kernel trained on observations at a timescale \( \rightarrow \) it will attempt to match smoothness at that timescale.
RQ excerpts that I found useful for discussion

How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

$$C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad \text{vs} \quad C_{LEG}(\tau) = B \exp\left(-\frac{\tau}{2} \left( N N^\top + R - R^\top \right)\right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top$$

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- ...but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel

This apparent contradiction is resolved by considering the different timescales involved in any Gaussian Process:

- LEG uniformly approximate any stationary covariance $\rightarrow$ uniformly accurate predictions at any fixed timescale
- LEG kernel trained on observations at a timescale $\rightarrow$ it will attempt to match smoothness at that timescale.
- Previous figure: LEG trained on observations at a timescale of .1, so they match the covariance at that scale.
How well do LEG kernels approximate the Rational Quadratic (RQ) kernel?

\[ C_{RQ}(\tau) = \frac{2}{1 + \tau^2} \quad vs \quad C_{LEG}(\tau) = B \exp\left( -\frac{\tau}{2} \left( NN^\top + R - R^\top \right) \right) B^\top + \delta_{\tau=0} \Lambda \Lambda^\top \]

RQ is profoundly different than LEG kernel, yet every RQ kernel is arbitrarily close to a LEG kernel:

- RQ spectrum decays exponentially, and LEG spectrum is asymptotically an inverse polynomial
- ...but this paper guarantees an RQ kernel can be matched uniformly well by a LEG kernel

This apparent contradiction is resolved by considering the different timescales involved in any Gaussian Process:

- LEG uniformly approximate any stationary covariance → uniformly accurate predictions at any fixed timescale
- LEG kernel trained on observations at a timescale → it will attempt to match smoothness at that timescale.
- Previous figure: LEG trained on observations at a timescale of .1, so they match the covariance at that scale.
\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s)ds + Ndw(s) \right) \]

\[ x(t) | z(t) \sim \mathcal{N} \left( Bz(t), \Lambda\Lambda^\top \right) \]
\[ z(t) = z(0) + \int_0^t \left( -\frac{1}{2} \left( NN^\top + R - R^\top \right) z(s) ds + Ndw(s) \right) \]

\[ x(t) | z(t) \sim \mathcal{N} \left( Bz(t), \Lambda \Lambda^\top \right) \]

- rank 5 ($\ell = 5$) LEG seems to do as well as Chapter 5.4 of Rasmussen and Williams and similar.
- leggps is a usable and very fast package for working with LEG. See also celerite.
gaussian processes and state-space models

latent exponentially generated gaussian processes (leggp)

toolkit: cyclic reduction for parallelizing message passing in tridiagonal (gauss markov) models
Reminder: SSM/GMP/LEGGP are linear time

(I’m making $z_t$ discrete time for simplicity of the reminder here, but it’s the same in c.t.)

all linear operations maintain joint gaussianity

In case this inference operation is distant in your mind:

$$p_\theta(z_t|z_{t-1}) = \mathcal{N}(Az_{t-1}, Q)$$

$$p_\theta(x_t|z_t) = \mathcal{N}(Bz_t, \Lambda)$$

$$p_\theta(z|x) \propto \prod_{t=1}^{T} p_\theta(x_t|z_t)p_\theta(z_t|z_{t-1}) = \mathcal{N}(\mu, \Sigma)$$

$$z = [z_1, \ldots, z_T]^\top \in \mathbb{R}^{dT} \quad x = [x_1, \ldots, x_T]^\top \in \mathbb{R}^{pT}$$
Considering this problem in its natural form clarifies all:

\[ \mathcal{N}(\mu, \Sigma) \propto \exp \left\{ -\frac{1}{2} (z - \mu)^\top \Sigma^{-1} (z - \mu) \right\} \propto \exp \left\{ \left[ \frac{1}{2} \Sigma^{-1} \mu \right]^\top \left[ \begin{array}{c} z \\ zz^\top \end{array} \right] \right\} \overset{\Delta}{=} \exp \left\{ \left[ h \right]^\top \left[ \begin{array}{c} z \\ zz^\top \end{array} \right] \right\} \]
Considering this problem in its natural form clarifies all:

\[ \mathcal{N}(\mu, \Sigma) \propto \exp \left\{ -\frac{1}{2} (\mathbf{z} - \mu)^\top \Sigma^{-1} (\mathbf{z} - \mu) \right\} \propto \exp \left\{ \left[ \begin{array}{c} \Sigma^{-1} \mu \\ -\frac{1}{2} \Sigma^{-1} \end{array} \right]^\top \left[ \begin{array}{c} \mathbf{z} \\ \mathbf{z}^\top \end{array} \right] \right\} \triangleq \exp \left\{ \left[ \begin{array}{c} \mathbf{h} \\ \mathbf{J} \end{array} \right]^\top \left[ \begin{array}{c} \mathbf{z} \\ \mathbf{z}^\top \end{array} \right] \right\} \]

Now the LDS model:

\[ p_{\theta}(\mathbf{z}|\mathbf{x}) \propto \prod_{t=1}^{T} p_{\theta}(x_t|z_t)p_{\theta}(z_t|z_{t-1}) \]
Considering this problem in its natural form clarifies all:

\[
\mathcal{N}(\mu, \Sigma) \propto \exp\left\{ -\frac{1}{2} (\mathbf{z} - \mu)^\top \Sigma^{-1} (\mathbf{z} - \mu) \right\} \propto \exp\left\{ \begin{bmatrix} \Sigma^{-1} \mu \\ -\frac{1}{2} \Sigma^{-1} \end{bmatrix}^\top \begin{bmatrix} \mathbf{z} \\ \mathbf{z}^\top \end{bmatrix} \right\} \triangleq \exp\left\{ \begin{bmatrix} \mathbf{h} \\ \mathbf{J} \end{bmatrix}^\top \begin{bmatrix} \mathbf{z} \\ \mathbf{z}^\top \end{bmatrix} \right\}
\]

Now the LDS model:

\[
p_\theta(\mathbf{z}|\mathbf{x}) \propto \prod_{t=1}^{T} p_\theta(x_t|z_t)p_\theta(z_t|z_{t-1})
\]

\[
= \prod_{t=1}^{T} \exp\left\{ -\frac{1}{2} (x_t - Bz_t)^\top \Lambda^{-1} (x_t - Bz_t) - \frac{1}{2} (z_t - Az_{t-1})^\top Q^{-1} (z_t - Az_{t-1}) \right\}
\]
Considering this problem in its natural form clarifies all:

\[ N(\mu, \Sigma) \propto \exp \left\{ -\frac{1}{2} (z - \mu) \Sigma^{-1} (z - \mu) \right\} \propto \exp \left\{ \left[ \begin{array}{c} \Sigma^{-1} \mu \\ -\frac{1}{2} \Sigma^{-1} \end{array} \right]^\top \left[ \begin{array}{c} z \\ \Sigma^{-1} \end{array} \right] \right\} \triangleq \exp \left\{ [h]^\top [z] \right\} \]

Now the LDS model:

\[
p_\theta(z|x) \propto \prod_{t=1}^T p_\theta(x_t|z_t) p_\theta(z_t|z_{t-1}) \\
= \prod_{t=1}^T \exp \left\{ -\frac{1}{2} (x_t - Bz_t)^\top \Lambda^{-1} (x_t - Bz_t) - \frac{1}{2} (z_t - Az_{t-1})^\top Q^{-1} (z_t - Az_{t-1}) \right\} \\
\propto \prod_{t=1}^T \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top (Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B) z_t \right\}
\]

(with some laziness around \( t = 0 \))
Considering this problem in its natural form clarifies all:

\[
\mathcal{N}(\mu, \Sigma) \propto \exp \left\{ -\frac{1}{2} (z - \mu)^\top \Sigma^{-1} (z - \mu) \right\} \propto \exp \left\{ \left[ \Sigma^{-1} \mu \right]^\top \left[ \begin{array}{c} z \\ z \end{array} \right] \right\} \overset{\Delta}{=} \exp \left\{ \left[ h \right]^\top \left[ \begin{array}{c} z \\ z \end{array} \right] \right\}
\]

Now the LDS model:

\[
p_{\theta}(z|x) \propto \prod_{t=1}^{T} p_{\theta}(x_t|z_t)p_{\theta}(z_t|z_{t-1}) = \prod_{t=1}^{T} \exp \left\{ -\frac{1}{2} (x_t - B z_t)^\top \Lambda^{-1} (x_t - B z_t) - \frac{1}{2} (z_t - A z_{t-1})^\top Q^{-1} (z_t - A z_{t-1}) \right\}
\]

\[
= \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top (Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B) z_t \right\}
\]

(with some laziness around \( t = 0 \))

So it is immediate that the natural parameter \( h = [h_1, \ldots, h_T]^\top \) has form:

\[
h_t = B^\top \Lambda^{-1} x_t
\]
Considering this problem in its natural form clarifies all:

\[
\mathcal{N}(\mu, \Sigma) \propto \exp \left\{ -\frac{1}{2} (\mathbf{z} - \mu)^\top \Sigma^{-1} (\mathbf{z} - \mu) \right\} \propto \exp \left\{ \left[ \frac{\Sigma^{-1} \mu}{-\frac{1}{2} \Sigma^{-1}} \right]^\top \left[ \frac{\mathbf{z}}{zz^\top} \right] \right\} \triangleq \exp \left\{ [h]^\top \left[ \frac{\mathbf{z}}{zz^\top} \right] \right\}
\]

Now the LDS model:

\[
p_\theta(\mathbf{z}|\mathbf{x}) \propto \prod_{t=1}^{T} p_\theta(x_t|z_t)p_\theta(z_t|z_{t-1})
\]

\[
= \prod_{t=1}^{T} \exp \left\{ -\frac{1}{2} (x_t - Bz_t)^\top \Lambda^{-1} (x_t - Bz_t) - \frac{1}{2} (z_t - A z_{t-1})^\top Q^{-1} (z_t - A z_{t-1}) \right\}
\]

\[
\propto \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top \left( Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B \right) z_t \right\}
\]

(with some laziness around \( t = 0 \))

So it is immediate that the natural parameter \( \mathbf{h} = [h_1, ..., h_T]^\top \) has form:

\[
h_t = B^\top \Lambda^{-1} x_t
\]
...and the natural parameter $J$:

$$p_\theta(z|x) \propto \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top (B^\top \Lambda^{-1} B + Q^{-1} + A^\top Q^{-1} A) z_t \right\}$$
...and the natural parameter $J$:

$$p_\theta(z|x) \propto \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top \left( B^\top \Lambda^{-1} B + Q^{-1} + A^\top Q^{-1} A \right) z_t \right\}$$

$$-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^\top \\ F_0 & D_1 & F_1^\top \\ & \ddots & \ddots \\ & & F_{T-1}^\top & D_{T-1} \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} D_t = Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B \\ F_t = -2Q^{-1} A \end{bmatrix}$$
...and the natural parameter $J$:

$$p_{\theta}(z|x) \propto \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top (B^\top \Lambda^{-1} B + Q^{-1} + A^\top Q^{-1} A) z_t \right\}$$

$$-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^\top & & & \\ F_0 & D_1 & F_1^\top & & \\ & \ddots & \ddots & \ddots & \\ & & F_{T-1}^\top & & \\ & & & D_T & \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} D_t = Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B \\ F_t = -2Q^{-1} A \end{bmatrix}$$

So what?

- block tridiagonal precision matrix: $z_t^\top z_{t-\tau}$ terms $= 0 \ \forall \tau > 1$. 
...and the natural parameter $J$:

$$p_\theta(z|x) \propto \prod_{t=1}^{T} \exp \left\{ x_t^\top \left( \Lambda^{-1} B \right) z_t + z_t^\top \left( Q^{-1} A \right) z_{t-1} - \frac{1}{2} z_t^\top \left( B^\top \Lambda^{-1} B + Q^{-1} + A^\top Q^{-1} A \right) z_t \right\}$$

$$-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^\top & & & \\ F_0 & D_1 & F_1^\top & & \\ & \ddots & \ddots & \ddots & \\ & & F_{T-1}^\top & \ddots & \\ & & & F_{T-1} & D_T \end{bmatrix}$$

where

$$D_t = Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B$$

$$F_t = -2Q^{-1} A$$

So what?

- **block tridiagonal precision matrix**: $z_t z_{t-\tau}^\top$ terms $= 0 \ \forall \tau > 1$.
- Recover $\mu = -\frac{1}{2} J^{-1} h$ and $\Sigma = -\frac{1}{2} J^{-1}$.
...and the natural parameter $J$:

$$p_\theta(z|x) \propto \prod_{t=1}^{T} \exp \left\{ x_t^\top (\Lambda^{-1} B) z_t + z_t^\top (Q^{-1} A) z_{t-1} - \frac{1}{2} z_t^\top \left( B^\top \Lambda^{-1} B + Q^{-1} + A^\top Q^{-1} A \right) z_t \right\}$$

$$- \frac{1}{2} J = \begin{bmatrix} D_0 & F_0^\top & F_1^\top & \cdots & F_{T-1}^\top \\ F_0 & D_1 & F_1 & \cdots & F_{T-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ F_{T-1} & \cdots & F_{T-1} & D_T \end{bmatrix}$$

where

$$D_t = Q^{-1} + A^\top Q^{-1} A + B^\top \Lambda^{-1} B$$

$$F_t = -2Q^{-1}A$$

So what?

- **block tridiagonal precision matrix:** $z_t z_{t-\tau}^\top$ terms $= 0 \ \forall \tau > 1$.
- Recover $\mu = -\frac{1}{2} J^{-1} h$ and $\Sigma = -\frac{1}{2} J^{-1}$
- Reparameterize $z^\ell = -\frac{1}{2} J^{-1} h + \left( -\frac{1}{2} J^{-1} \right)^{\frac{1}{2}} \epsilon^\ell$
Kalman filter/smoother

Still, so what...

$$\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^T & F_1^T & \cdots & F_{T-1}^T \\ F_0 & D_1 & F_1^T & \cdots & F_{T-1}^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F_{T-1} & F_{T-1}^T & D_{T-1} & \cdots & D_T \end{bmatrix} = \begin{bmatrix} L_0 & 0 & 0 & \cdots & 0 \\ C_0 & L_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{T-1} & 0 & C_T & \cdots & 0 \\ 0 & 0 & C_{T-1} & \cdots & L_T \end{bmatrix} \triangleq RR^T$$
Still, so what...

\[-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^T & \cdots & F_T^T \\ F_0 & D_1 & \cdots & F_{T-1}^T \\ \vdots & \vdots & \ddots & \vdots \\ F_{T-1} & \cdots & F_T^T & D_T \end{bmatrix} = \begin{bmatrix} L_0 & 0 & \cdots & 0 \\ C_0 & L_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C_{T-1} & \cdots & 0 & L_T \end{bmatrix} \begin{bmatrix} L_0 & C_0^T & \cdots & C_{T-1}^T \\ 0 & L_1 & \cdots & C_{T-1}^T \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_T \end{bmatrix} \triangleq RR^T\]

Iterative solve:

\[L_0 L_0^T = D_0 , \ C_0 L_0^T = F_0 , \ C_0 C_0^T + L_1 L_1^T = D_1 , \ \ldots , \ C_{T-1} C_{T-1}^T + L_T L_T^T = D_T\]
Kalman filter/smoother

Still, so what...

\[-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^T & F_1^T & \cdots & F_{T-1}^T \\ F_0 & D_1 & F_1^T & \cdots & F_{T-1}^T \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ F_{T-1} & D_T \end{bmatrix} = \begin{bmatrix} L_0 & 0 & 0 & \cdots & 0 \\ C_0 & L_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ C_{T-2} & \cdots & C_{T-1} & 0 & 0 \\ 0 & \cdots & 0 & L_T \end{bmatrix} \begin{bmatrix} L_0 & 0 & 0 & \cdots & 0 \\ C_0^T & L_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ C_{T-1}^T & \cdots & C_{T-1}^T & 0 & 0 \\ 0 & \cdots & 0 & L_T^T \end{bmatrix} \triangleq RR^T \]

Iterative solve:

\[L_0L_0^T = D_0, \quad C_0L_0^T = F_0, \quad C_0C_0^T + L_1L_1^T = D_1, \quad \cdots, \quad C_{T-1}C_{T-1}^T + L_TL_T^T = D_T\]

Fast Cholesky factorization (aka fwd/bk substitution, message passing, KF/KS, etc.)

https://software.intel.com/en-us/node/531896
Kalman filter/smoother

Still, so what...

\[-\frac{1}{2} J = \begin{bmatrix} D_0 & F_0^\top & F_1^\top & \cdots & F_{T-1}^\top \\ F_0 & D_1 & F_1 & \cdots & F_{T-1} \\ & \ddots & \ddots & \ddots & \vdots \\ & & F_{T-1} & D_T & \end{bmatrix} = \begin{bmatrix} L_0 & 0 & 0 & \cdots & 0 \\ C_0 & L_1 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \vdots \\ & & & C_{T-1} & L_T \\ & & & & 0 \end{bmatrix} \begin{bmatrix} L_0 & C_0^\top & \cdots & C_{T-1}^\top \\ 0 & L_1 & \cdots & C_{T-1}^\top \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & L_T \end{bmatrix} \triangleq RR^\top \]

Iterative solve:

\[L_0 L_0^\top = D_0, \quad C_0 L_0^\top = F_0, \quad C_0 C_0^\top + L_1 L_1^\top = D_1, \quad \ldots, \quad C_{T-1} C_{T-1}^\top + L_T L_T^\top = D_T \]

Fast Cholesky factorization (aka fwd/bk substitution, message passing, KF/KS, etc.)

And finally we recover either the reparameterization or the closed form posterior:

\[z^\ell = R^{-1} R^{-\top} h + R^{-\top} \epsilon^\ell \]

\[p_\theta(z|x) = \mathcal{N} \left( R^{-1} R^{-\top} h, R^{-1} R^{-\top} \right) \]

▶ All key operations are \( \mathcal{O}(T) \).
Is linear good enough? Not these days:

- Conventional KF/RTS smoothing is linear but entirely serial (forward/backward)
- No access to GPU parallelization

Numerical linear algebra to our rescue (again):

"Cyclic Reduction has proved ... very powerful for solving structured matrix problems. In particular for matrices which are (block) Toeplitz and (block) tri-diagonal, the method is especially useful. The basic idea is to eliminate half the unknowns, regroup the equations and again eliminate half the unknowns. The process is continued ad nauseam."

- Gene Golub (Gander and Golub 1997)

Start with a tridiagonal system:

following Bini and Meini 2008
CYCLIC REDUCTION

Is linear good enough? Not these days:

- Conventional KF/RTS smoothing is linear but entirely serial (forward/backward)
- No access to GPU parallelization
- Numerical linear algebra to our rescue (again):

"Cyclic Reduction has proved ... very powerful for solving structured matrix problems. In particular for matrices which are (block) Toeplitz and (block) tri-diagonal, the method is especially useful. The basic idea is to eliminate half the unknowns, regroup the equations and again eliminate half the unknowns. The process is continued ad nauseum."

- Gene Golub (Gander and Golub 1997)

Start with a tridiagonal system:

following Bini and Meini 2008
Is linear good enough? Not these days:

- Conventional KF/RTS smoothing is linear but entirely serial (forward/backward)
- No access to GPU parallelization
- Numerical linear algebra to our rescue (again):

"Cyclic Reduction has proved ... very powerful for solving structured matrix problems. In particular for matrices which are (block) Toeplitz and (block) tri-diagonal, the method is especially useful. The basic idea is to eliminate half the unknowns, regroup the equations and again eliminate half the unknowns. The process is continued ad nauseum."

- Gene Golub (Gander and Golub 1997)
CYCLIC REDUCTION

Is linear good enough? Not these days:

- Conventional KF/RTS smoothing is linear but entirely serial (forward/backward)
- No access to GPU parallelization
- Numerical linear algebra to our rescue (again):

"Cyclic Reduction has proved ... very powerful for solving structured matrix problems. In particular for matrices which are (block) Toeplitz and (block) tri-diagonal, the method is especially useful. The basic idea is to eliminate half the unknowns, regroup the equations and again eliminate half the unknowns. The process is continued ad nauseum."

- Gene Golub (Gander and Golub 1997)

Start with a tridiagonal system:

\[
\begin{bmatrix}
A & C & 0 \\
B & A & \cdots \\
& \ddots & C \\
0 & B & A \\
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2 \\
\vdots \\
u_n \\
\end{bmatrix} =
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n \\
\end{bmatrix}
\]

following Bini and Meini 2008
Subdivide (odds and evens) and permute the system:

\[
\begin{bmatrix}
A & 0 & C & 0 \\
\vdots & B & \ddots & \vdots \\
0 & A & 0 & B \\
B & C & 0 & A \\
\vdots & \vdots & \ddots & \vdots \\
0 & B & C & 0 & A
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_3 \\
\vdots \\
u_{2^{q-1}-1} \\
u_2 \\
u_4 \\
\vdots \\
u_{2^{q-1}-2}
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_3 \\
\vdots \\
b_{2^{q-1}-1} \\
b_2 \\
b_4 \\
\vdots \\
b_{2^{q-1}-2}
\end{bmatrix}
\]

Write in Schur form:

\[
\begin{bmatrix}
H_{11} & H_{12} \\
H_{21} & H_{22}
\end{bmatrix}
\begin{bmatrix}
\mathbf{u}_{\text{odd}} \\
\mathbf{u}_{\text{even}}
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{b}_{\text{odd}} \\
\mathbf{b}_{\text{even}}
\end{bmatrix}
\]

\((H_{22} - H_{21}H_{11}^{-1}H_{12})\mathbf{u}_{\text{even}} = \mathbf{b}^{(1)}\)

\(\mathbf{b}^{(1)} = \mathbf{b}_{\text{even}} - H_{21}H_{11}^{-1}\mathbf{b}_{\text{odd}}\)
Subdivide (odds and evens) and permute the system:

\[
\begin{bmatrix}
A & 0 & C & 0 \\
\vdots & & B & \ddots \\
0 & \ddots & A & 0 \\
B & C & 0 & A
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_3 \\
\vdots \\
u_{2^q-1} \\
u_{2^q-2} \\
\vdots \\
u_{2^q-2}
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_3 \\
\vdots \\
b_{2^q-1} \\
b_{2^q-2} \\
\vdots \\
b_{2^q-2}
\end{bmatrix}
\]

Write in Schur form:

\[
\begin{bmatrix}
H_{11} & H_{12} \\
H_{21} & H_{22}
\end{bmatrix}
\begin{bmatrix}
u_{\text{odd}} \\
u_{\text{even}}
\end{bmatrix}
= 
\begin{bmatrix}
\begin{bmatrix}
u_{\text{odd}} \\
u_{\text{even}}
\end{bmatrix}
\end{bmatrix}
\]

\[(H_{22} - H_{21}H_{11}^{-1}H_{12})u_{\text{even}} = b^{(1)} \]

\[b^{(1)} = b_{\text{even}} - H_{21}H_{11}^{-1}b_{\text{odd}}\]

"Magically, the Schur complement has the same structure as the original matrix"...
“Magically, the Schur complement has the same structure as the original matrix”...

\[
\begin{bmatrix}
A^{(1)} & C^{(1)} & 0 \\
B^{(1)} & A^{(1)} & \ddots \\
& \ddots & \ddots & C^{(1)} \\
0 & B^{(1)} & A^{(1)} \\
\end{bmatrix}
\begin{bmatrix}
u_2 \\
u_4 \\
\vdots \\
u_{2^q-2} \\
\end{bmatrix} =
\begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
\vdots \\
b_{2^q-1}^{(1)} \\
\end{bmatrix}
\]

\[b_i^{(1)} = b_{2i} - BA^{-1}b_{2i-1} - CA^{-1}b_{2i+1}, \quad i = 1, \ldots, 2^{q-1} - 1\]

and

\[A^{(1)} = A - BA^{-1}C - CA^{-1}B\]
\[B^{(1)} = -BA^{-1}B\]
\[C^{(1)} = -CA^{-1}C\]

Subdivide again (and again... in fact up to \(k = \log_2 T\) times) to produce a sequence of smaller tridiag systems:

\[
\begin{bmatrix}
A^{(k)} & C^{(k)} & 0 \\
B^{(k)} & A^{(k)} & \ddots \\
& \ddots & \ddots & C^{(k)} \\
0 & B^{(k)} & A^{(k)} \\
\end{bmatrix}
\begin{bmatrix}
u_{1,2^k} \\
u_{2,2^k} \\
\vdots \\
u_{(2^{q-k}-1)2^k} \\
\end{bmatrix} =
\begin{bmatrix}
b_1^{(k)} \\
b_2^{(k)} \\
\vdots \\
b_{2^q-k-1}^{(k)} \\
\end{bmatrix}
\]

How to achieve parallelization:
“Magically, the Schur complement has the same structure as the original matrix”...

\[
\begin{bmatrix}
A^{(1)} & C^{(1)} & 0 \\
B^{(1)} & A^{(1)} & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
0 & B^{(1)} & A^{(1)} & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
u_2 \\
u_4 \\
\vdots \\
u_{2^q-2}
\end{bmatrix}
=
\begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
\vdots \\
b_{2^q-1-1}^{(1)}
\end{bmatrix}
\quad \text{and}

b_i^{(1)} = b_{2i} - BA^{-1}b_{2i-1} - CA^{-1}b_{2i+1}, \ i = 1, \ldots, 2^{q-1} - 1
\]

Subdivide again (and again... in fact up to \( k = \log_2 T \) times) to produce a sequence of smaller tridiag systems:

\[
\begin{bmatrix}
A^{(k)} & C^{(k)} & 0 \\
B^{(k)} & A^{(k)} & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
0 & B^{(k)} & A^{(k)} & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
u_{1.2^k} \\
u_{2.2^k} \\
\vdots \\
u_{(2^q-k)-1.2^k}
\end{bmatrix}
=
\begin{bmatrix}
b_1^{(k)} \\
b_2^{(k)} \\
\vdots \\
b_{2^q-k-1}^{(k)}
\end{bmatrix}
\]

How to achieve parallelization:

- Original approach: solve the last subdivided systems, back substitute to compute all remaining unknowns.
“Magically, the Schur complement has the same structure as the original matrix”...

\[
\begin{bmatrix}
A^{(1)} & C^{(1)} & 0 \\
B^{(1)} & A^{(1)} & \ddots \\
\vdots & \ddots & \ddots & C^{(1)} \\
0 & B^{(1)} & \cdots & A^{(1)}
\end{bmatrix}
\begin{bmatrix}
u_2 \\
u_4 \\
\vdots \\
u_{2q-2}
\end{bmatrix} =
\begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
\vdots \\
b_{2q-1-1}^{(1)}
\end{bmatrix}
\]

\[b_i^{(1)} = b_{2i} - BA^{-1}b_{2i-1} - CA^{-1}b_{2i+1}, \quad i = 1, \ldots, 2^{q-1} - 1\]

Subdivide again (and again... in fact up to \(k = \log_2 T\) times) to produce a sequence of smaller tridiag systems:

\[
\begin{bmatrix}
A^{(k)} & C^{(k)} & 0 \\
B^{(k)} & A^{(k)} & \ddots \\
\vdots & \ddots & \ddots & C^{(k)} \\
0 & B^{(k)} & \cdots & A^{(k)}
\end{bmatrix}
\begin{bmatrix}
u_{1,2^k} \\
u_{2,2^k} \\
\vdots \\
u_{(2^{q-k}-1)2^k}
\end{bmatrix} =
\begin{bmatrix}
b_1^{(k)} \\
b_2^{(k)} \\
\vdots \\
b_{2^{q-k}-1}^{(k)}
\end{bmatrix}
\]

How to achieve parallelization:

- Original approach: solve the last subdivided systems, back substitute to compute all remaining unknowns.
- Sweet (1988) recasts \(A^{(k)}u = b\) as \(u = \sum c_i(A - d_iI)^{-1}b\), resulting in \(2^k\) entirely parallel system solves.
Easy way out: leggps package (on github)
Easy way out: leggps package (on github)

- fully parallel construction using Sweet
CYCLIC REDUCTION

Easy way out: leggps package (on github)

- fully parallel construction using Sweet
- handles irregularly-spaced time points
Easy way out: leggps package (on github)

- fully parallel construction using Sweet
- handles irregularly-spaced time points
- handles vector-valued observations
Easy way out: leggps package (on github)

- fully parallel construction using Sweet
- handles irregularly-spaced time points
- handles vector-valued observations
- no parameters to tune (good use of BFGS given small parameterization)
Easy way out: leggps package (on github)

- fully parallel construction using Sweet
- handles irregularly-spaced time points
- handles vector-valued observations
- no parameters to tune (good use of BFGS given small parameterization)

Challenges:
CYCLIC REDUCTION

Easy way out: leggps package (on github)

▶ fully parallel construction using Sweet

▶ handles irregularly-spaced time points

▶ handles vector-valued observations

▶ no parameters to tune (good use of BFGS given small parameterization)

Challenges:

▶ we should do a deep dive on pivoted cholesky, multigrid methods, and cyclic reduction
Easy way out: leggps package (on github)

- fully parallel construction using Sweet
- handles irregularly-spaced time points
- handles vector-valued observations
- no parameters to tune (good use of BFGS given small parameterization)

Challenges:

- we should do a deep dive on pivoted cholesky, multigrid methods, and cyclic reduction
- what is CR from a gaussian conditioning perspective (Schur complementation is the core op)
gaussian processes and state-space models

latent exponentially generated gaussian processes (leggp)

toolkit: cyclic reduction for parallelizing message passing in tridiagonal (gauss markov) models