Assimilation of large-scale data sets
Alex Bibov, Heikki Haario
Contents

- Prediction problem
- Data assimilation at glance
- Least-Squares and generalizations
- Kalman filter and approximations
- Toy case: wind motion
- Toy case: shallow water
- Remark: CUDA — supercomputer at your lap
- Toy case: river flood
- Toy case: dam break
- Toy case: sea waves
Most significant science problem is prediction
Prediction of what happens “tomorrow” requires knowledge of what happens “today”
Most natural phenomena are chaotic
Chaotic systems are sensitive to “the butterfly effect”
Prior knowledge is never available at exact precision
Chaotic trajectory

Small initial variations

Lead to huge differences in feature behavior
Chaotic behavior: Lorenz-3 model
Chaotic behavior: Lorenz-3 model

- The water wheel behavior is described by the following ODE system:
  \[
  \begin{align*}
  \dot{x} &= \sigma(y - x), \\
  \dot{y} &= x(\rho - z) - y, \\
  \dot{z} &= xy - \beta z.
  \end{align*}
  \]

- Parameter values \(\sigma = 10, \beta = 8/3, \rho = 28\) are known to cause chaotic behavior.
Chaotic behavior: Lorenz-3 model

Lorenz Attractor

three near-identical initial points diverge into chaos
Examples of chaotic processes

- Wind motion
- Ocean behavior
- Climate change
- Controlled complex systems
- Financial markets
Applications of the “tamed chaos”

• Weather forecast
• Optimizing wind turbines
• Natural disaster prediction
• Assisted system construction (e.g. cruise control)
• Military applications (e.g. target tracking and locking)
• Economy fall prediction
Chaotic systems: food for thought

• What other chaotic systems can you imagine?
• Have you encountered chaotic systems in your studies/research?
• For any chaotic process that you have in mind try to perform some experiments: use the same starting conditions in each experiment and get sure that even though the starting conditions appear same, each time the process behaves completely differently
Data assimilation at glance

• Assume that considered system is modeled by a discrete time evolution operator:

\[ s_{k+1} = M_k(s_k), \]

where \( s_k \) is vector of parameters that fully describe the system.

• In the case of Lorenz-3 \( s_k = (x(t_k), y(t_k), z(t_k)) \).

• Assume that at each time point we observe the system in accordance with the following observation model:

\[ o_{k+1} = H_{k+1}(s_{k+1}). \]

• Data assimilation problem: how to combine \( s_{k+1} \) and \( o_{k+1} \) together.
Least squares: naïve approach

• We need to find an estimate \( \hat{s}_{k+1} \) of the state \( s_{k+1} \) benefiting from observed data \( o_{k+1} \).

• The estimate \( \hat{s}_{k+1} \) should be as close as possible to the “simulated value”.

• The estimate \( \hat{s}_{k+1} \) should minimize discrepancy with observed data \( o_{k+1} \).

**Minimize the following cost function:**

\[
F(\hat{s}_{k+1}) = \| \hat{s}_{k+1} - s_{k+1} \|^2 + \| o_{k+1} - H_{k+1}(\hat{s}_{k+1}) \|^2
\]

**NOTE:** \( H_{k+1} \) could be replaced with linearization \( H_{k+1}^{TL} = \frac{\partial H_{k+1}}{\partial x} \).
Naïve approach: drawbacks

- Evolution model $\mathcal{M}_k(s_k)$ is always an approximation of the natural phenomena. Let us introduce prediction error $\varepsilon_k \sim \mathcal{N}(0, C_{\varepsilon_k})$. Thence:
  $$s_{k+1} = \mathcal{M}_k(s_k) + \varepsilon_k.$$

- Observations are read by devices that have certain measurement error. Consider observation error $\eta_{k+1} \sim \mathcal{N}(0, C_{\eta_{k+1}})$. Observation model is updated as follows:
  $$o_{k+1} = \mathcal{H}_{k+1}(s_{k+1}) + \eta_{k+1}.$$

- We need to decide on relative contribution of prediction $s_{k+1}$ and observation $o_{k+1}$ to the estimate $\hat{s}_{k+1}$. 
Least squares: generalization

- Introduce notation for “weighted” length $\|x\|_C = x^T C x$.
- Evolution operator $\mathcal{M}_k$ can be to some extent approximated by its Jacobian $M_k^{TL} = \frac{\partial \mathcal{M}_k}{\partial x}$.
- Compute approximate covariance matrix for random variable $\mathcal{M}_k(x_k)$:
  \[ C_{k+1}^p = M_k^{TL} C_k (M_k^{TL})^T + C_{\varepsilon_k}, \text{ where } C_k = \text{cov}(x_k, x_k). \]
- If $x$ is a random vector variable and $\text{cov}(x, x) = C$, then length of $x$ with respect to relative “importance” of each component can be computed as $\|x\|_{C^{-1}}$. 
Kalman filter

- Reformulate previously defined least-square fit using scaling by covariance data
- Scaling factor for observation discrepancy is given by covariance $C_{\eta_{k+1}}$
- Scaling factor for prediction discrepancy is given by covariance matrix $C_{k+1}^p$

Minimize the following cost function:

$$F(\hat{s}_{k+1}) = \|\hat{s}_{k+1} - s_{k+1}\|^2_{(C_{k+1}^p)^{-1}} + \|o_{k+1} - H_{k+1}(\hat{s}_{k+1})\|^2_{C_{\eta_{k+1}}^{-1}}$$

- Minimizer of $F(\hat{s}_{k+1})$ is called the Kalman filter estimate. Covariance matrix $C_{k+1}$ of this minimizing vector is given by inverse Hessian matrix of $F(\hat{s}_{k+1})$
Kalman filter: food for thought

Minimize the following cost function:

\[ F(\hat{s}_{k+1}) = \|\hat{s}_{k+1} - s_{k+1}\|^2_{(C^p_{k+1})^{-1}} + \|o_{k+1} - \mathcal{H}_{k+1}(\hat{s}_{k+1})\|^2_{C\eta^{-1}_{k+1}} \]

- Cost function \( F(\hat{s}_{k+1}) \) can be rewritten in the following form:
  \[ F(\hat{s}_{k+1}) = \hat{s}^T_{k+1} A \hat{s}_{k+1} + \hat{s}^T_{k+1} b + c, \]
  where \( A \) and \( b \) are positive-definite matrix and column-vector of appropriate dimensions; \( c \) is some constant.
- It can be proved that \( \nabla F(\hat{s}_{k+1}) = A \hat{s}_{k+1} + b. \)
- Therefore, since \( A \) is positive-definite, cost function \( F(\hat{s}_{k+1}) \) is minimized by solution to the following system of linear equations:
  \[ A \hat{s}_{k+1} = b. \]
- In other words \( \hat{s}_{k+1} = A^{-1} b \)
- Moreover, \( \text{cov}(\hat{s}_{k+1}, \hat{s}_{k+1}) = A^{-1} \text{cov}(b, b) A^{-1} \)
Kalman filter: food for thought

Minimize the following cost function:
\[
F(\hat{s}_{k+1}) = \|\hat{s}_{k+1} - s_{k+1}\|^2_{(C_{k+1}^{-1})} + \|o_{k+1} - H_{k+1}(\hat{s}_{k+1})\|^2_{C^{-1}_{\eta_{k+1}}}
\]

• By using argumentation from the previous slide derive explicit formulas for Kalman filter estimation:
  1. Reduce cost function \( F(\hat{s}_{k+1}) \) to canonical quadratic form:
     \[
     F(\hat{s}_{k+1}) = \hat{s}_{k+1}^TA\hat{s}_{k+1} + \hat{s}_{k+1}^Tb + c.
     \]
  2. Use formula \( \hat{s}_{k+1}^T = A^{-1}b \) to compute Kalman filter estimate
  3. Use formula \( cov(\hat{s}_{k+1}, \hat{s}_{k+1}) = A^{-1}cov(b, b)A^{-1} \)
  4. Prove, that in fact \( cov(\hat{s}_{k+1}, \hat{s}_{k+1}) = A^{-1} \)

• Can formulae of the same type be similarly derived for usual naïve least-squares.

• If yes, compare the formulae for Kalman filter and for naïve least-squares. When do they become equivalent?
Kalman filter: drawbacks

- Consider problem of predicting wind motion
- Among other parameters, one would need to model atmospheric pressure
- Assume that the model covers 10km-by-10km region and simulates atmospheric pressure at resolution of 5m
- State of such model comprises $4 \cdot 10^6$ elements
Kalman filter: drawbacks

- As we stated, the Kalman filter estimate requires storage of covariance data for the model state.
- Hence, the covariance matrix contains $16 \cdot 10^{12}$ elements.
- For double-precision computations, this would require about 116,4TBs of RAM storage (computer with the world’s biggest shared memory storage has only around 32TBs).
- Inversion of this covariance would require about $4 \cdot 10^{39}$ double-precision multiplications.
- Such inversion would require $7,5 \cdot 10^{15}$ years of computations in the world’s fastest supercomputer Tianhe-2 (Guangzhou, China).
Approximate Kalman filter

- Kalman filter becomes impracticable when model dimension grows
- Idea: many large-scale processes have significantly less “acting” degrees of freedom compared to dimension of their state space
- The “acting” parameters may interact in a highly non-linear manner, which is difficult to reveal
- A low-memory approximation of the classical Kalman filter can be used in general situation regardless of the model
Ensemble Kalman filter

- The simplest way to approximate problematic covariance matrices is to perform statistical sampling
- Assume that $s_k$ is known state.
- Perturb this state by normally distributed noise:
  $$\tilde{s}_k^i \sim \mathcal{N}(s_k, C_k), \ i = 1,2,\ldots,m$$
- Propagate perturbed states using the model:
  $$\tilde{s}_{k+1}^i = \mathcal{M}_k(\tilde{s}_k^i), \quad s_{k+1}^p = \mathcal{M}_k(s_k)$$
- Compute sampled covariance
  $$C_{k+1}^p = \frac{(\tilde{s}_{k+1} - s_k^p)(\tilde{s}_{k+1} - s_k^p)^T}{\sqrt{N}} + C_{\varepsilon_k}$$
Test case: two-layer qg-mode

- Simulates quasi-geostrophic ("slow") wind motion
- Resides on cylindrical surface vertically divided in two layers
- The boundary conditions on top and the bottom of the cylinder are fixed constants
- Governing equations with respect to unknown stream function $\psi_i(x, y)$

$$
q_1 = \nabla^2 \psi_1 - F_1(\psi_1 - \psi_2) + \beta y, \\
q_2 = \nabla^2 \psi_2 - F_2(\psi_2 - \psi_1) + \beta y + R_s, \\
\frac{D_1 q_1}{D t} = \frac{D_2 q_2}{D t} = 0,
$$

where $R_s = R_s(x, y)$ is orography surface,

$$
\frac{D_i}{D t} = \frac{\partial}{\partial t} + u_i \frac{\partial}{\partial x} + v_i \frac{\partial}{\partial y} \text{ and } \nabla \psi_i = (v_i, -u_i).$
$$
The QG-model: geometry

Layer interaction interface

Top Layer

Bottom Layer

Topography
The QG-model: example run
The QG-model: chaotic behavior
QG-model: Kalman filter benchmark
2D Saint-Venant System

\[
\begin{cases}
  h_t + (hu)_x + (hv)_y = 0, \\
  (hu)_t + \left( hu^2 + \frac{1}{2}gh^2 \right)_x + (huv)_y = -ghB_x, \\
  (hu)_t + (huv)_x + \left( hu^2 + \frac{1}{2}gh^2 \right)_y = -ghB_y,
\end{cases}
\]

- where \( h \) is water elevation above the bottom, \((u, v)^T\) is velocity vector, \( B \) is the bottom surface and \( g \) is acceleration of gravity.
- Denote: \( w(x, y) := h(x, y) + B(x, y) \) — water surface, \( U = (w, hu, hv)^T \).
- Saint-Venant system in vector form: \( U_t + F(U, B)_x + G(U, B)_y = S(U, B) \).
Discretization step: finite volumes
GPU-based implementation: CUDA

- The main idea behind CUDA: modern Graphical Processing Units process each color of a high-resolution display in parallel “almost” independently from each other.
- The calculations behind the final pixel color can be fairly complicated, although much simpler than those made by modern programs running on CPU.
- Therefore, GPUs can be thought of as massive-parallel computers comprising thousands of cores.
CUDA grid topology

- In most general case, CUDA grid is a three dimensional matrix consisted of so-called blocks.
- Each block is a three dimensional matrix itself, but composed of threads, where each thread is responsible of a command sequence execution.
- NOTE: threads located within single block CAN communicate to each other using high-performance register memory of the GPU.
- Threads within a single block are subdivided into “warps” each containing 32 threads. Each “warp” is a small vector machine, that is an SIMD processor, which means that commands located within single warp are physically executed in parallel.
CUDA grid topology

CUDA grid of blocks

CUDA single block of threads
CUDA: application to SWE model

Thread\( (j,k) \)

\[
\begin{align*}
U_{j,k} & \quad U_{j+1,k} \\
\vdots & \quad \vdots \\
U_{j,k+1} & \quad U_{j+1,k+1}
\end{align*}
\]

Thread\( (j,k+1) \)

Thread\( (j+1,k) \)

Thread\( (j+1,k+1) \)
CUDA: comparison run

- Experiment: a “Gaussian hill” of water falls down into a valley
- Domain is discretized by 256x256 staggered grid
- State vector has 196608 components
- Boundaries are modeled by reflecting walls
Toy case: river flood
Toy case: dam break
Toy case: sea waves