Probabilistic numerics at scale by distributed inference

Alex Ledbetter, Mykola Lukashchuk & Wouter Kouw

International Conference on Probabilistic Numerics 2026

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

https://github.com/biaslab/ProbNum2026-Tutorial

installation instructions

Pluto notebook
(julia)

Marimo notebook
(python)

TU/e

Outline

1. Problem statement
2. Classical numerical approach
3. Probabilistic numerical approach
4. Message passing approach
5. Stacking modular components

1. Problem specification

A Problem statement

B Heat transfer on a silicon die

C A system of linear equations

Keeping chips cool

This is a silicon die, specifically a Pentium II.

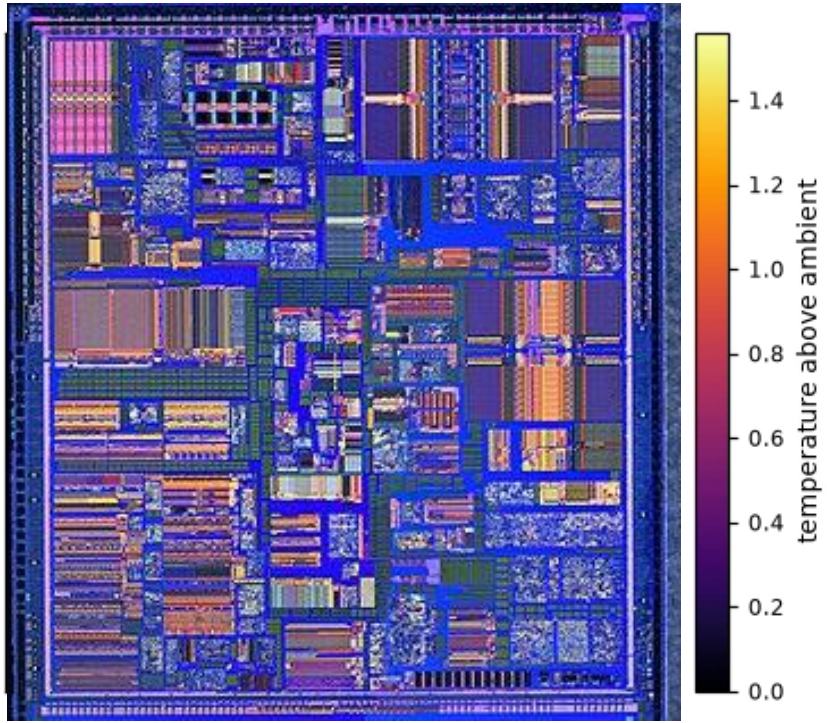
It contains a complete electronic circuit that performs a specific function (this one computes).

Each tile (region) draws power and heats up.

Heat spreads through the silicon material.

We want to keep the die cool, which we can do by "throttling" power consumption.

To decide which to "throttle", the control system needs the **steady-state temperature of every tile.**



Heat transfer on a silicon die

Let x_i be the temperature of the i-th tile, and b_i its power consumption.

The steady-state solution of the change in temperature over time is:

$$cx_i + \sum_{j \in \delta(i)} (x_i - x_j) = b_i$$

where $\delta(i)$ is the set of neighbors and c is a parameter.

The neighbor contributions can be approximated with the Laplacian Δx_i.

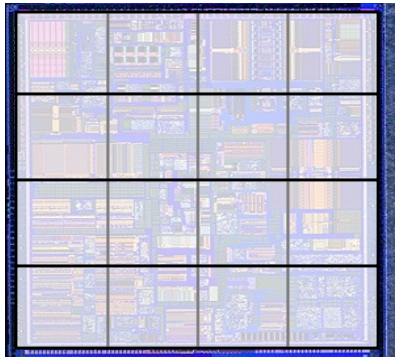
Then

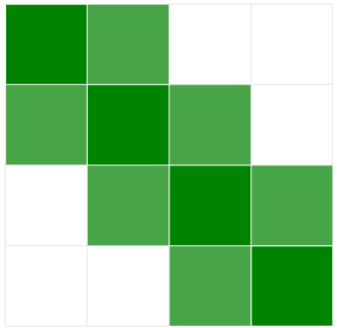
$$(c - \Delta)x_i = b_i$$

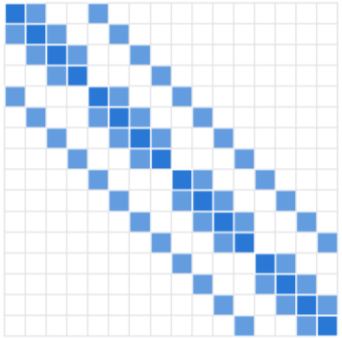
System of linear equations

Stack n temperatures and power consumptions into vectors $\boldsymbol{x} = [x_1 \dots x_n]$ and $\boldsymbol{b} = [b_1 \dots b_n]$.

This gives us a matrix $\boldsymbol{A} = (c\boldsymbol{I} - \boldsymbol{\Delta})$







So, the steady-state solution is $\boldsymbol{Ax} = \boldsymbol{b}$.

How can we infer unknown $\boldsymbol{x}$ from known $\boldsymbol{A}$ and $\boldsymbol{b}$?

2. Classical numerical approaches

A Jacobi method

B Gauss–Seidel method

C Conjugate gradient method

Jacobi method

First, $\boldsymbol{A}$ is split into diagonal and strictly triangular terms: $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}$. Then:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\ (\boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} &= \boldsymbol{b} \\ \boldsymbol{D}\boldsymbol{x} &= \boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} \\ \boldsymbol{x} &= \boldsymbol{D}^{-1}(\boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}) \end{aligned}$$

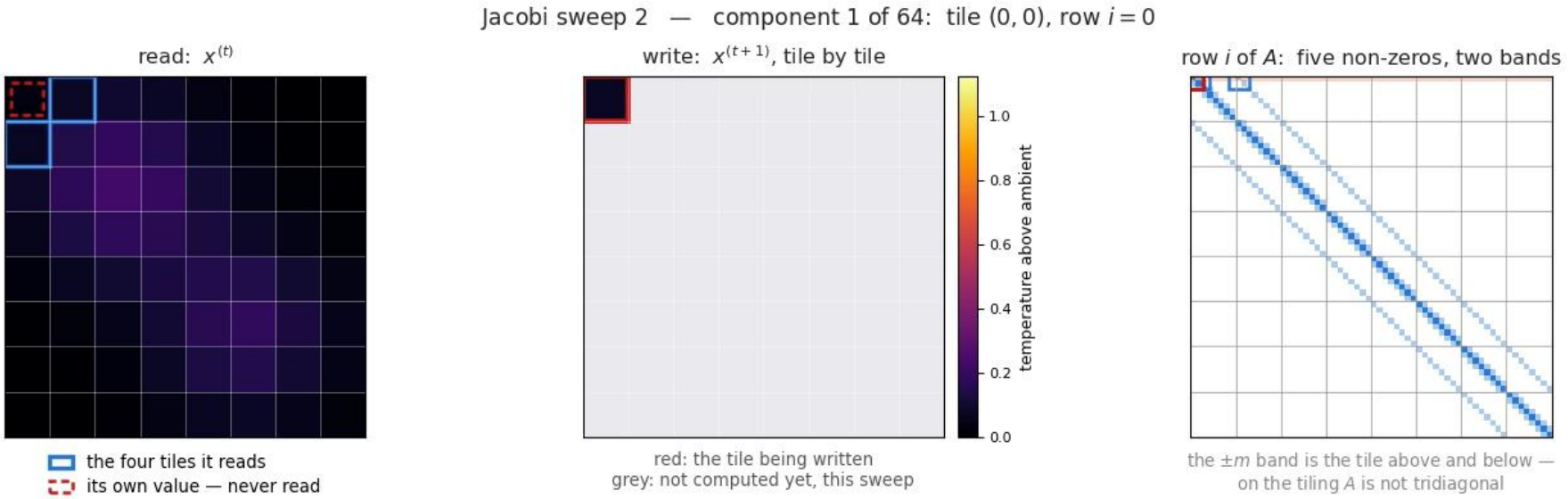
The fixed-point equation becomes a fixed-point iteration:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{D}^{-1}(\boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(t)})$$

In standard Jacobi iteration, you would update per equation, keeping all other equations fixed;

$$x_i^{(t+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(t)}\right)$$

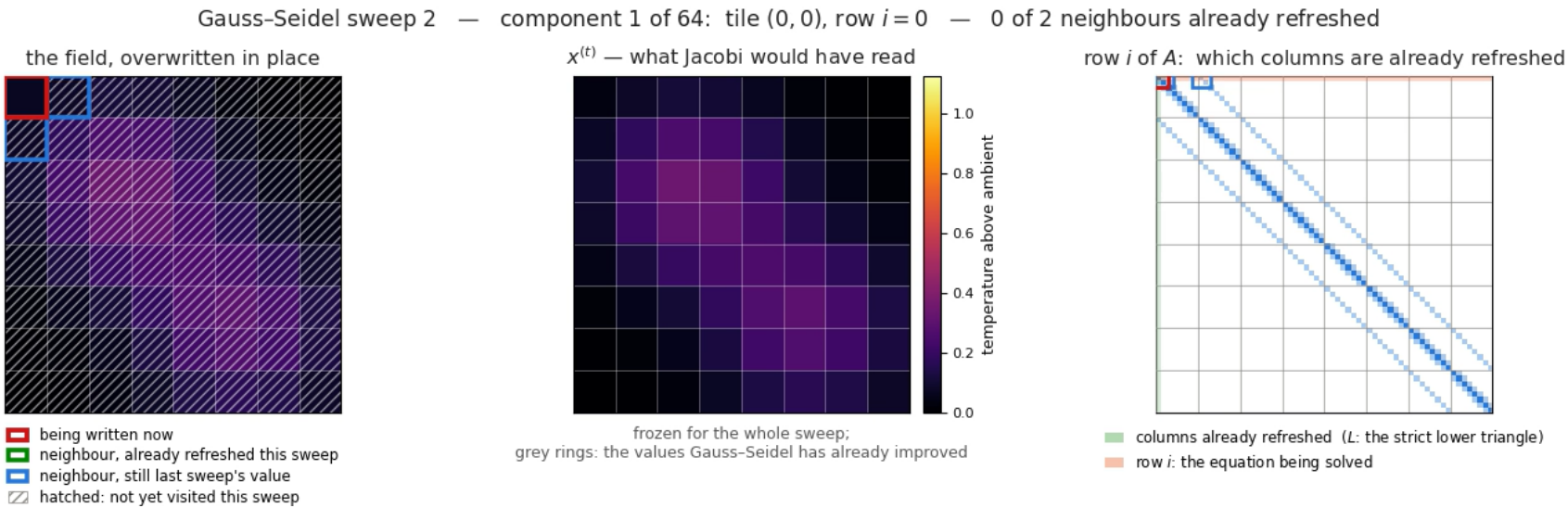
Jacobi method



Gauss-Seidel method

Gauss-Seidel is essentially the same but now you use the already computed values;

$$\boldsymbol{x}^{(t+1)} = (\boldsymbol{D} + \boldsymbol{L})^{-1}(\boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}^{(\mathrm{t})})$$



Conjugate gradient method

Consider the quadratic function $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{b}^T\boldsymbol{x}$.

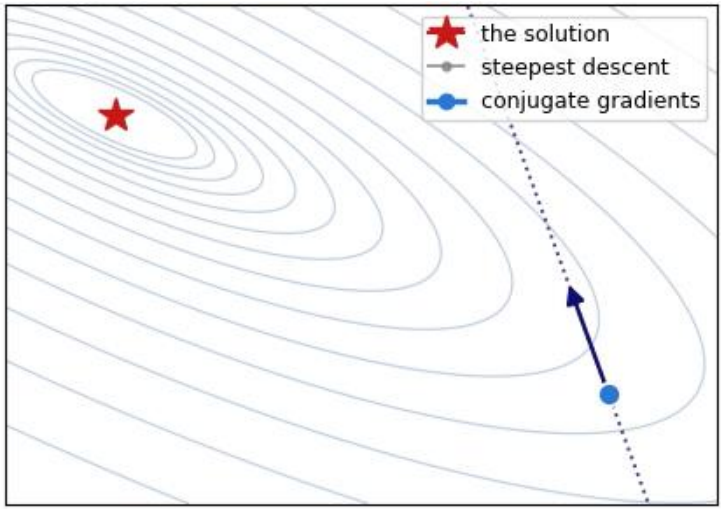
For symmetric positive definite A, $f(\boldsymbol{x})$ will be minimal when its derivative, $2\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{b}$, is 0.

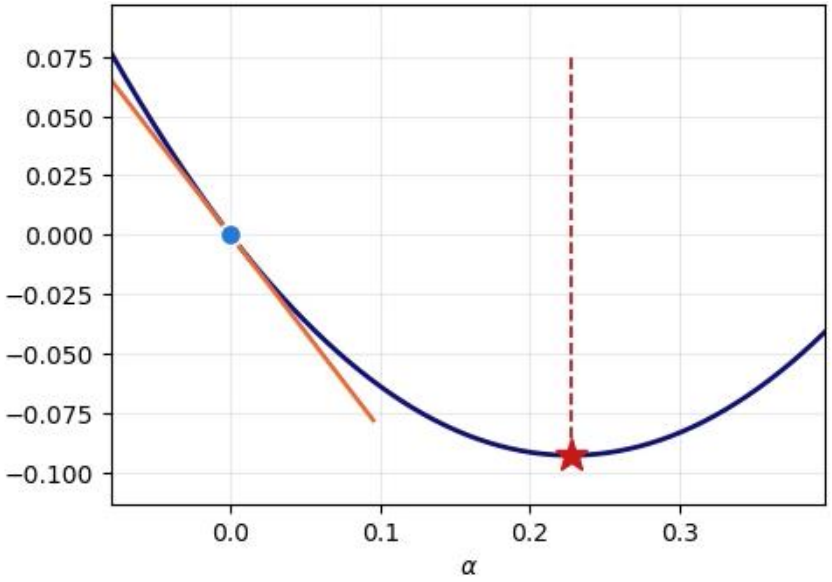
Algorithm:

- Start with an initial point $\boldsymbol{x}^{(0)}$ and initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$.
- Pick a direction $\boldsymbol{s}_0$ and compute step size: $\alpha_0 = \frac{\boldsymbol{s}_0^T \boldsymbol{r}^{(0)}}{\boldsymbol{s}_0^T \boldsymbol{A}\boldsymbol{s}_0}$
- Update point $\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + \alpha_0 \boldsymbol{s}_0$
- Update residual: $\boldsymbol{r}^{(1)} = \boldsymbol{r}^{(0)} - \alpha_0 \boldsymbol{A}\boldsymbol{s}_0$
- Repeat until n.

Conjugate gradient method

The directions are chosen to be conjugate with respect to $\boldsymbol{A}$, i.e., $\boldsymbol{s}_i^T \boldsymbol{A} \boldsymbol{s}_j = 0$ for $i \neq j$.

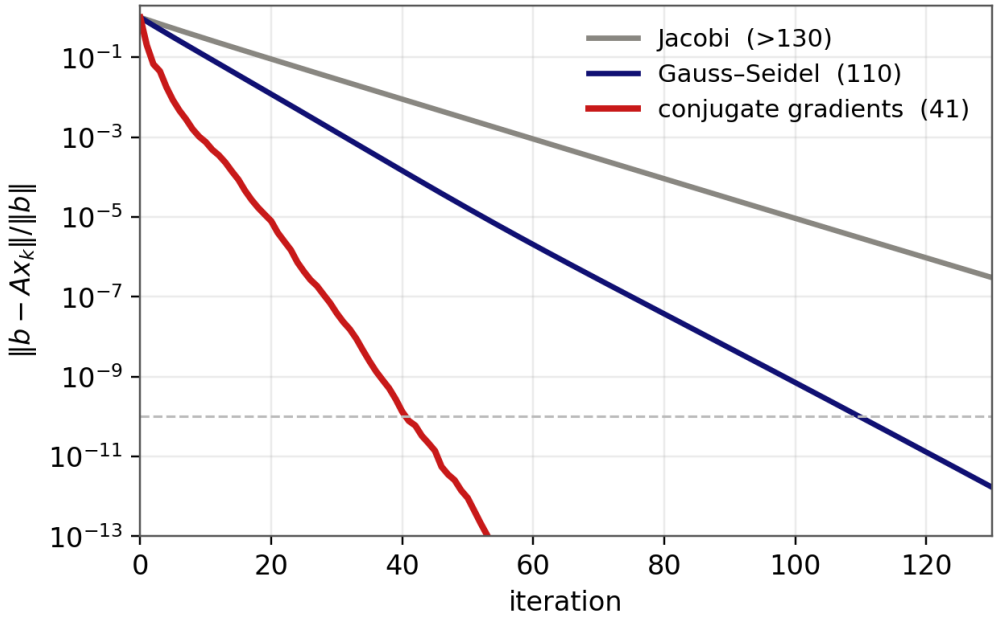




TU/e

Experiment

Consider a 24 x 24 grid tiling ($n = 576$) with $c = 0.4$.



TU/e

3. Probabilistic numerical approach

A BayesCG

B Uncertainty

TU/e

BayesCG

Put a prior distribution on the unknown: $p(\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{x}^{(0)}, \boldsymbol{\Sigma}^{(0)}\right)$.

Now pick m search directions $\boldsymbol{S} = [\boldsymbol{s}_1 \ \cdots \boldsymbol{s}_m]$.

Let $z_j = \boldsymbol{s}_j^T \boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{s}_j^T \boldsymbol{b}$ for $j = 1, \ldots m$.

This is a Dirac likelihood, $p\left(z_j \middle| \boldsymbol{x}\right) = \delta\left(z_j - \boldsymbol{s}_j^T \boldsymbol{A}\boldsymbol{x}\right)$.

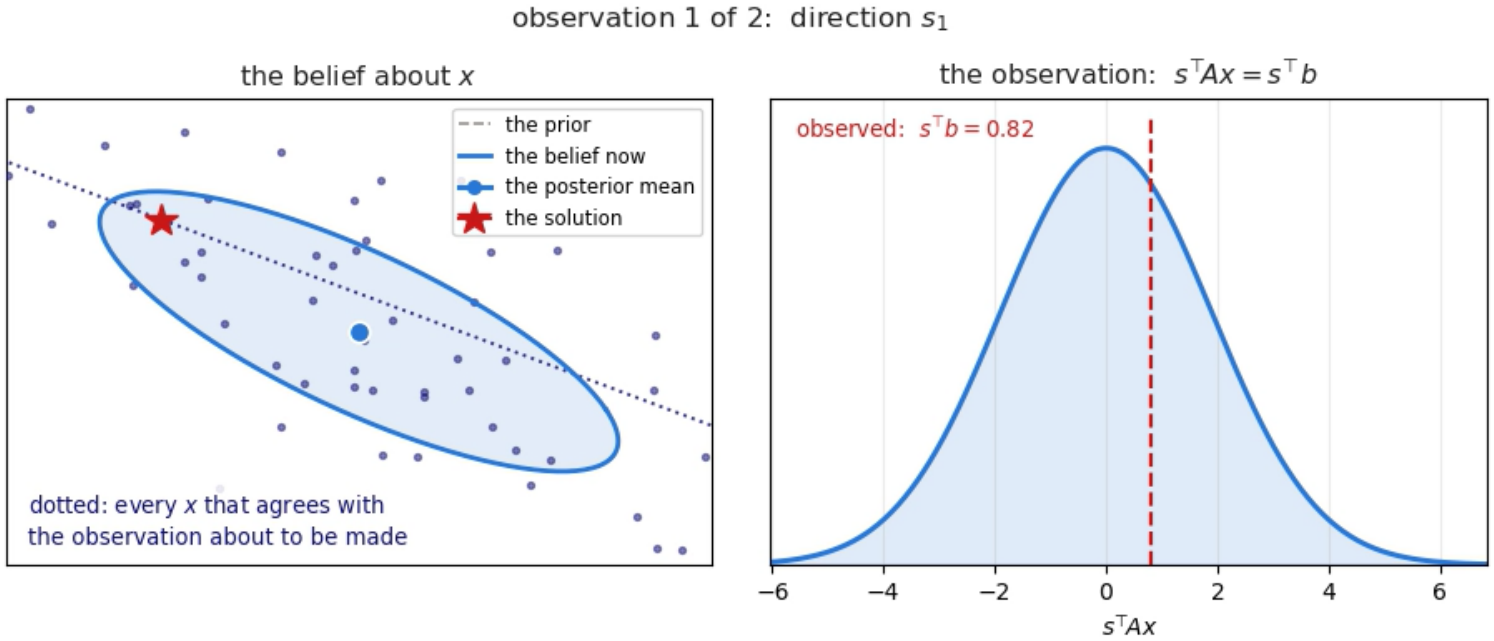
The posterior is then

$$p(\boldsymbol{x}|z) = \mathcal{N}\left(\boldsymbol{x}\middle|\boldsymbol{x}^{(m)}, \boldsymbol{\Sigma}^{(m)}\right)$$

whose parameters are an update to the prior's

$$\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(0)} + \boldsymbol{\Sigma}^{(0)}\boldsymbol{A}^T\boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{\Sigma}^{(0)}\boldsymbol{A}^T\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}\right)$$
$$\boldsymbol{\Sigma}^{(m)} = \boldsymbol{\Sigma}^{(0)} - \boldsymbol{\Sigma}^{(0)}\boldsymbol{A}^T\boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{\Sigma}^{(0)}\boldsymbol{A}^T\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{\Sigma}^{(0)}$$

BayesCG


observation 1 of 2: direction s_1
the belief about x
the prior
the belief now
the posterior mean
the solution
dotted: every x that agrees with
the observation about to be made
the observation: $s^\top Ax = s^\top b$
observed: $s^\top b = 0.82$
−6
−4
−2
0
2
4
6
$s^\top Ax$
the belief expects $s^\top Ax = 0.00 \pm 1.90$; the answer is 0.82

4. Message passing approach to $Ax = b$

A Setup

B Forney-style Factor Graphs

C Belief propagation

D Solving $Ax = b$ by Message-Passing on a Factor-Graph

Message-passing approach: Setup

$$A\mathbf{x} = \mathbf{b}$$

Goal: Solve Linear System

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right)$$

Multivariate Gaussian

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = 0$$

Mode

$$\mathbf{x} = A^{-1}\mathbf{b}$$

$\mathbf{x}$-configuration at mode

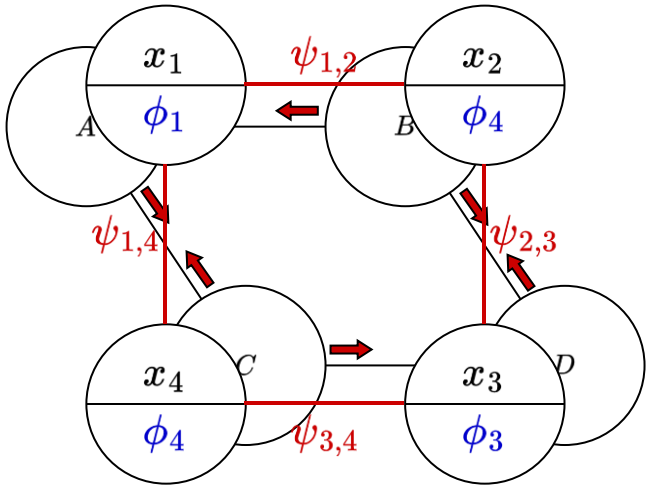
$$\mu = \Lambda^{-1}\eta$$

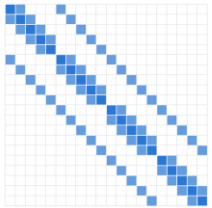
$\mathbf{x}$-configuration at mean

How to find the mode / mean?

TU/e

Message-passing approach: Setup





Structure of A

Gaussian Belief Propagation for Solving Systems of Linear Equations: Theory and Application

Ori Shental, Danny Bickson*, Paul H. Siegel, Jack K. Wolf and Danny Dolev

Abstract

The canonical problem of solving a system of linear equations arises in numerous contexts in information theory, communication theory and related fields. In this contribution, we develop a solution based upon Gaussian belief propagation (GaBP) that does not involve direct matrix inversion. The iterative nature of our approach allows for a distributed message-passing implementation of the solution algorithm. We address the properties of the GaBP solver, including convergence, exactness, computational complexity, message-passing efficiency and its relation to classical solution methods. We use numerical examples and applications, like linear detection, to illustrate these properties through the use of computer simulations. This empirical study demonstrates the attractiveness (*e.g.*, faster convergence rate) of the proposed GaBP solver in comparison to conventional linear-algebraic iterative solution methods.

Index Terms

Belief propagation, sum-product, system of linear equations, probabilistic inference, max-product, Gaussian elimination, iterative solution methods, Moore-Penrose pseudoinverse, linear detection, Poisson's equation.

The first two authors contributed equally to this work.

O. Shental, P. H. Siegel and J. K. Wolf are with the Center for Magnetic Recording Research, CMRR, University of California - San Diego (UCSD), 9500 Gilman Drive, La Jolla, CA 92093, USA (Email: {oshental,psiegel,jwolf}@ucsd.edu).

* Contact Author. D. Bickson and D. Dolev are with the School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem 91904, Israel (Email: {daniel51,dolev}@cs.huji.ac.il).

The material in this paper was presented in part at the forty-fifth Annual Allerton Conference on Communication, Control, and Computing, September 2007 and on the IEEE International Symposium on Information Theory (ISIT), July 2008.

Paper submitted to IEEE Transactions on Information Theory, November 1, 2018.

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right)$$

$$-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x} = \sum_i \left(b_i x_i - \frac{1}{2}A_{ii}x_i^2\right) - \sum_{i<j} A_{ij}x_i x_j$$

$$p(\mathbf{x}) = \prod_i \phi_i(x_i) \prod_{i,j} \psi_{ij}(x_i, x_j)$$

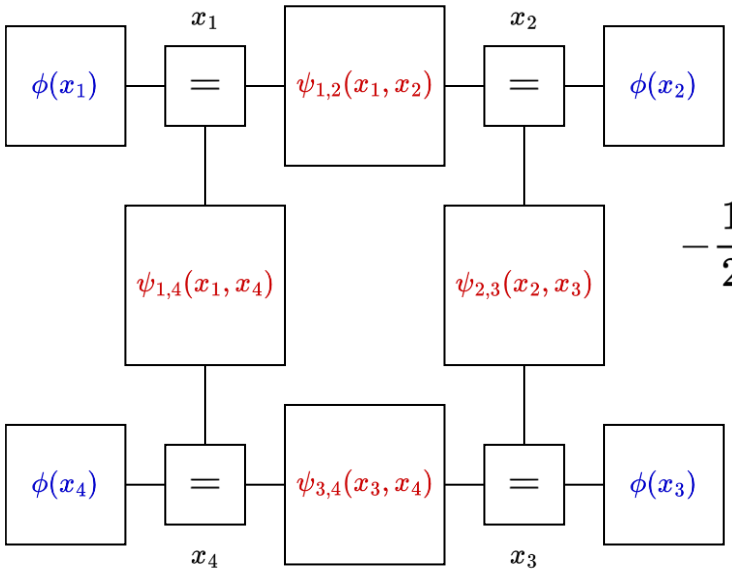
$$\phi_i(x_i) = \exp\left(b_i x_i - \frac{1}{2}A_{ii}x_i^2\right)$$

Unary

$$\psi_{ij}(x_i, x_j) = \exp\left(-A_{ij}x_i x_j\right)$$

Binary

Message-passing approach: Setup



Forney-style Factor Graph

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right)$$

$$-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x} = \sum_i \left(b_i x_i - \frac{1}{2}A_{ii}x_i^2\right) - \sum_{i<j} A_{ij}x_i x_j$$

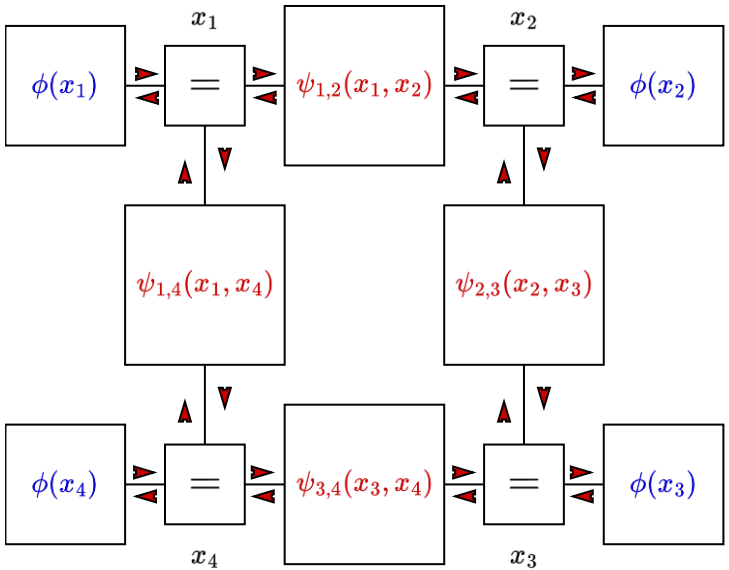
$$p(\mathbf{x}) = \prod_i \phi_i(x_i) \prod_{i<j} \psi_{ij}(x_i, x_j)$$

$$\phi_i(x_i) = \exp\left(b_i x_i - \frac{1}{2}A_{ii}x_i^2\right)$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-A_{ij}x_i x_j\right)$$

TU/e

Message-passing approach: Setup



Forney-style Factor Graph

- Message-passing is a scalable, distributed approach to approximate **marginal** inference.
- We infer $q(x_i)$ for each variable x_i in state vector $\mathbf{x}$.
- We then stack the means to obtain:

$$\mathbf{x} = A^{-1}\mathbf{b} = \begin{bmatrix} m\left(q(x_1)\right) \\ \vdots \\ m\left(q(x_N)\right) \end{bmatrix}$$

4. Message passing approach to $Ax = b$

A Setup

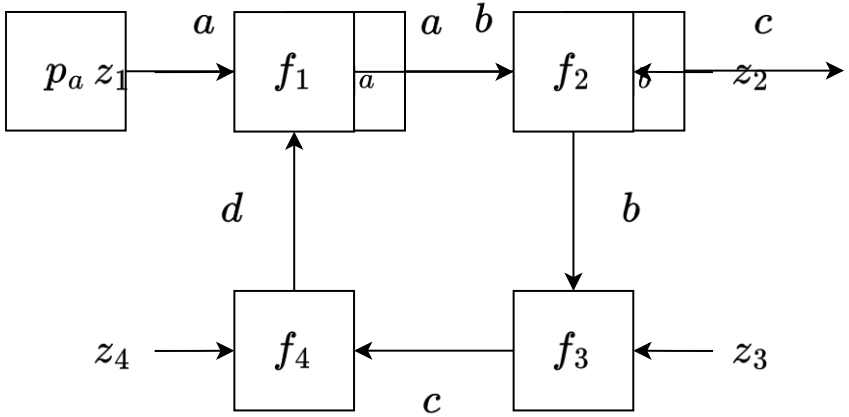
B Forney-style Factor Graphs

C Belief propagation

D Solving $Ax = b$ by Message-Passing on a Factor-Graph

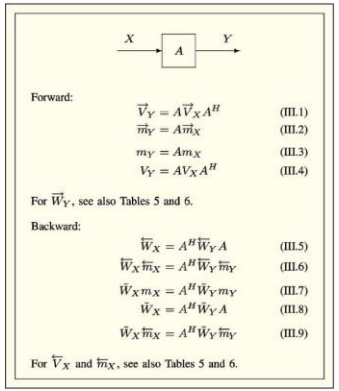
Message-passing approach: Forney-style Factor-graphs

$$p(a,b,c,d) \propto f_1(d,a,z_1)\, f_2(a,b,c)\, f_3(c,d,z_3)\, f_4(d,a,z_4)$$



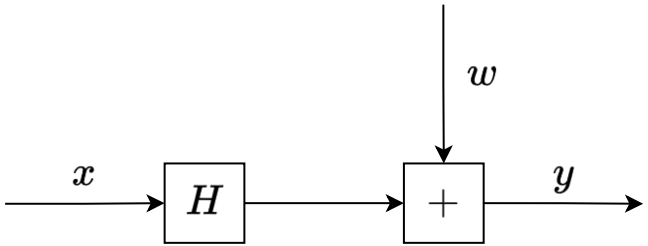
Factor Graph: Insightful visual representation of p(...)

Message-passing approach: Forney-style Factor-graphs



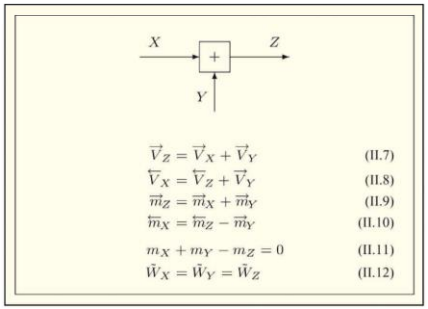
$$y = Hx + w$$

Linear factors



Multiplication node

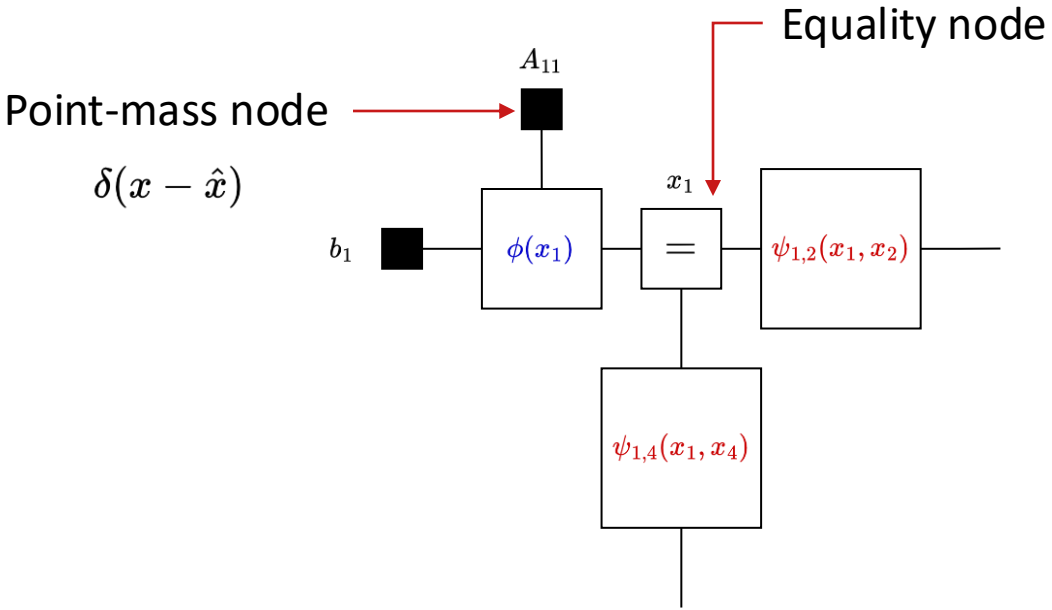
Summation node

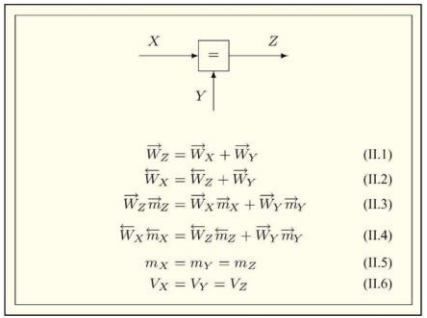


TU/e

Message-passing approach: Forney-style Factor-graphs

Other common factors

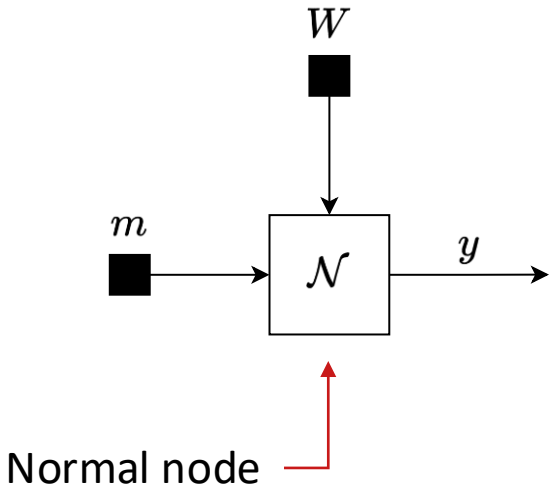




TU/e

Message-passing approach: Forney-style Factor-graphs

Other common factors

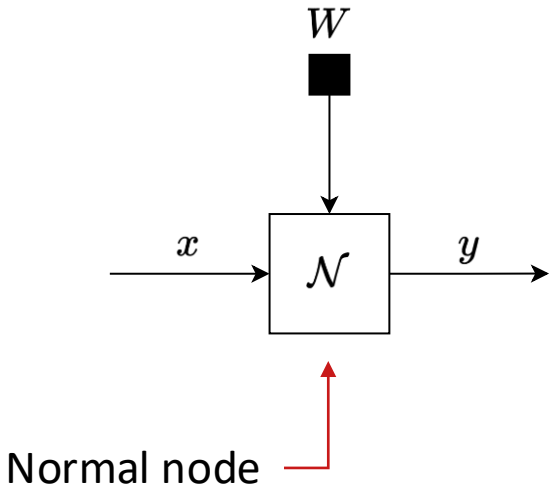


$$\vec{\mu}_{\mathcal{N}\to y}(y) = \mathcal{N}\left(y \mid m, W^{-1}\right)$$

TU/e

Message-passing approach: Forney-style Factor-graphs

Other common factors



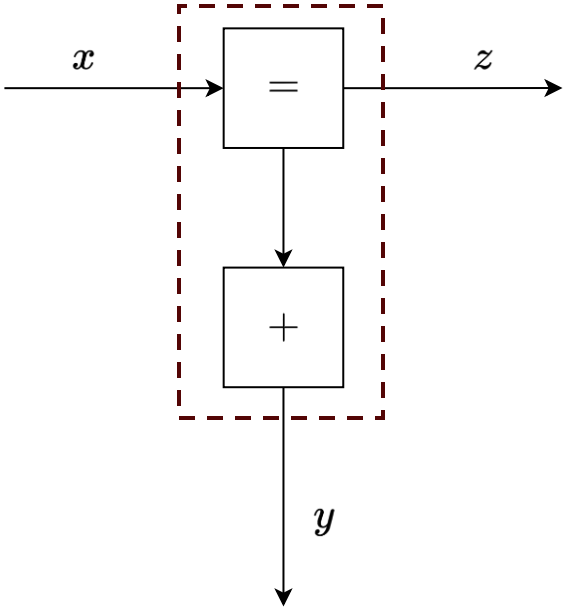
$$\vec{\mu}_{\mathcal{N}\to y}(y) = \mathcal{N}\left(y \mid m_x, V_x + W^{-1}\right)$$

$$\overleftarrow{\mu}_{\mathcal{N}\to x}(x) = \mathcal{N}\left(x \mid m_y, V_y + W^{-1}\right)$$

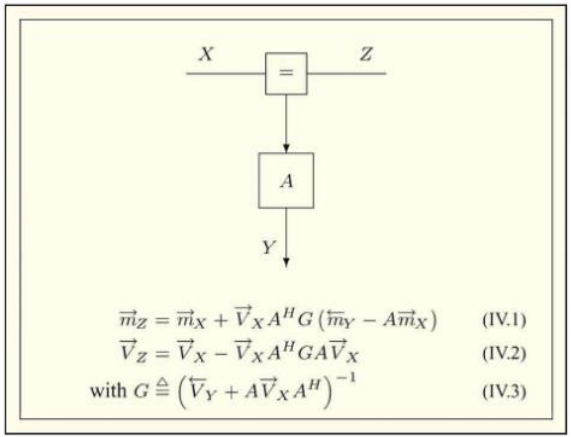
TU/e

Message-passing approach: Forney-style Factor-graphs

We can combine nodes into “composite nodes”



Node composition



4. Message passing approach to $Ax = b$

A Setup

B Forney-style Factor Graphs

C Belief propagation

D Solving $Ax = b$ by Message-Passing on a Factor-Graph

Message-passing approach: Belief-propagation

Great! So how do we perform message-passing inference?

We consider only Belief Propagation

aka Sum-Product message-passing

Three steps:

(1) Messages

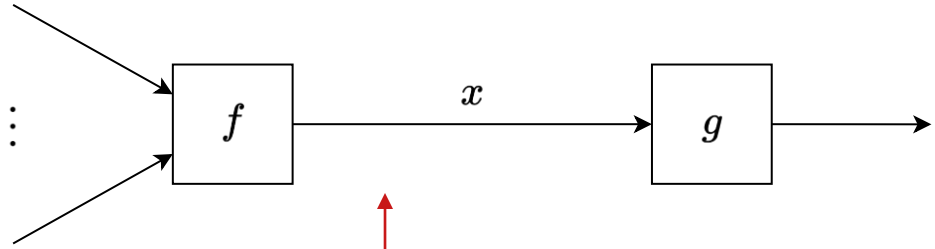
(2) Products

(3) Iteration

TU/e

Message-passing approach: Belief-propagation

(1) Messages



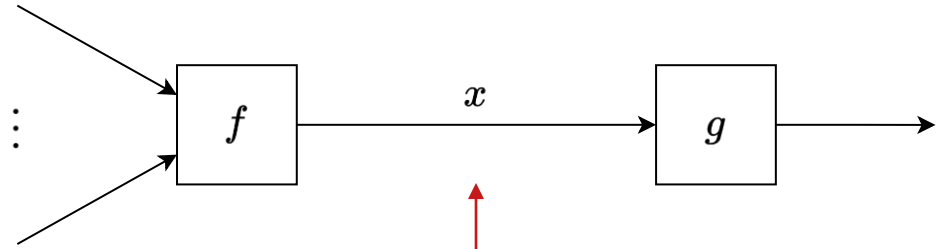
Incoming messages

Integral of product of factor and other incoming messages, over the other variables of f

BP message from f to x

Message-passing approach: Belief-propagation

(2) Products



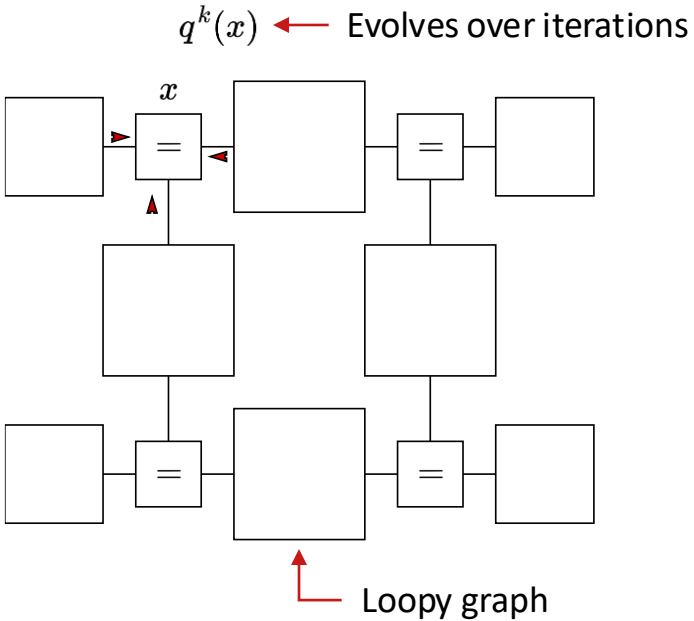
Marginal for x at iteration k, $q^k(x)$

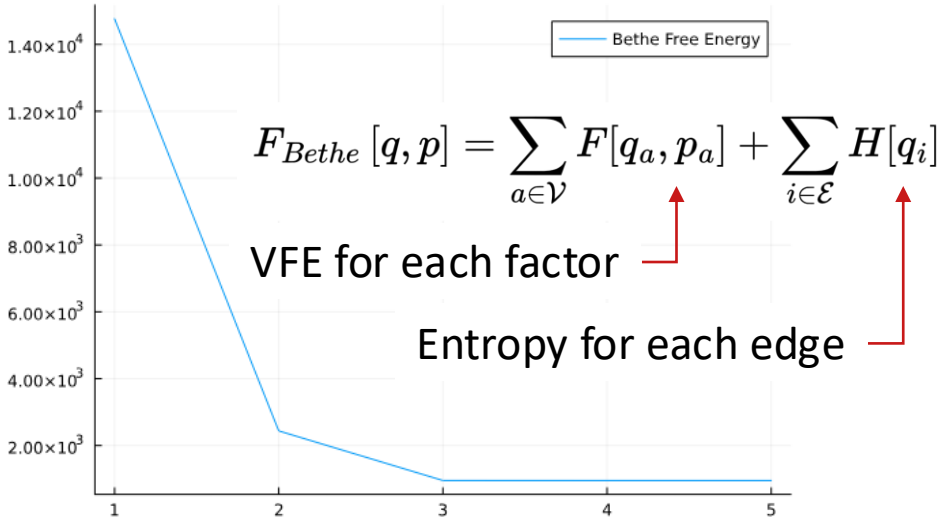
Formed as product of messages to x

TU/e

Message-passing approach: Belief-propagation

(3) Iteration





TU/e

4. Message passing approach to $Ax = b$

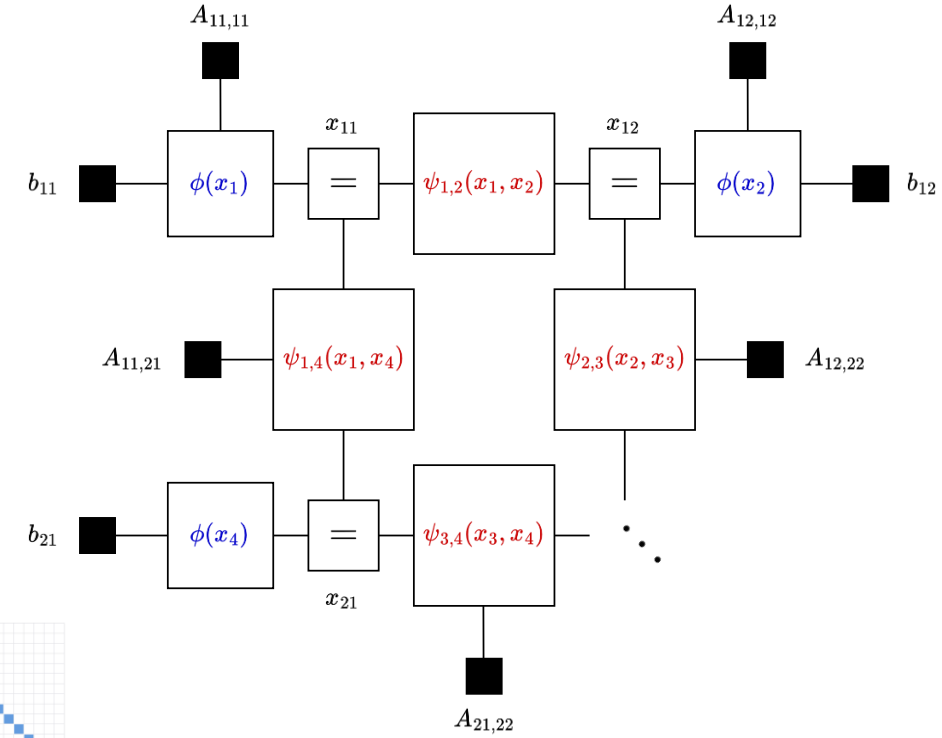
A Setup

B Forney-style Factor Graphs

C Belief propagation

D Solving $Ax = b$ by Message-Passing on a Factor-Graph

Message-passing approach: Solving $A\mathbf{x} = \mathbf{b}$



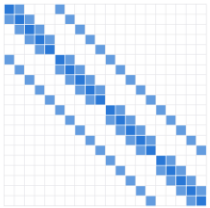
$$A\mathbf{x} = \mathbf{b}$$

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right)$$

$$p(\mathbf{x}) = \prod_i \phi_i(x_i) \prod_{i<j} \psi_{ij}(x_i, x_j)$$

$$\phi_i(x_i) = \exp\left(b_i x_i - \frac{1}{2} A_{ii} x_i^2\right)$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-A_{ij} x_i x_j\right)$$

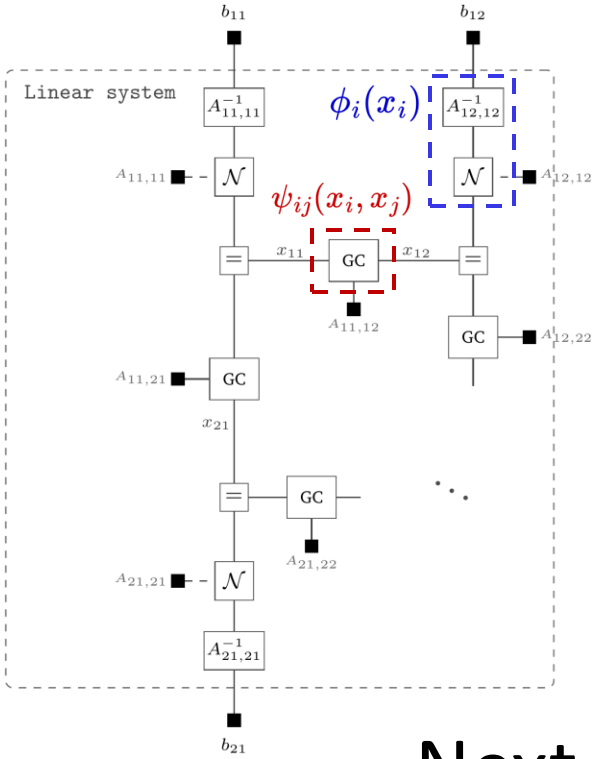


Structure of A

TU/e

Message-passing approach: Solving $A\mathbf{x} = \mathbf{b}$

Same graph,
different appearance



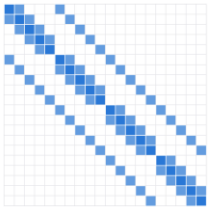
$$A\mathbf{x} = \mathbf{b}$$

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right)$$

$$p(\mathbf{x}) = \prod_i \phi_i(x_i) \prod_{i<j} \psi_{ij}(x_i, x_j)$$

$$\phi_i(x_i) = \exp\left(b_i x_i - \frac{1}{2} A_{ii} x_i^2\right)$$

$$\psi_{ij}(x_i, x_j) = \exp\left(-A_{ij} x_i x_j\right)$$



Structure of A

Next up: RxInfer!

TU/e

5. Stacking modular components

A Anylyzing GaBP (Bayesian Jacobi solver)

B Building

TU/e

5. Stacking modular components

A Anylyzing GaBP (Bayesian Jacobi solver)

B Building

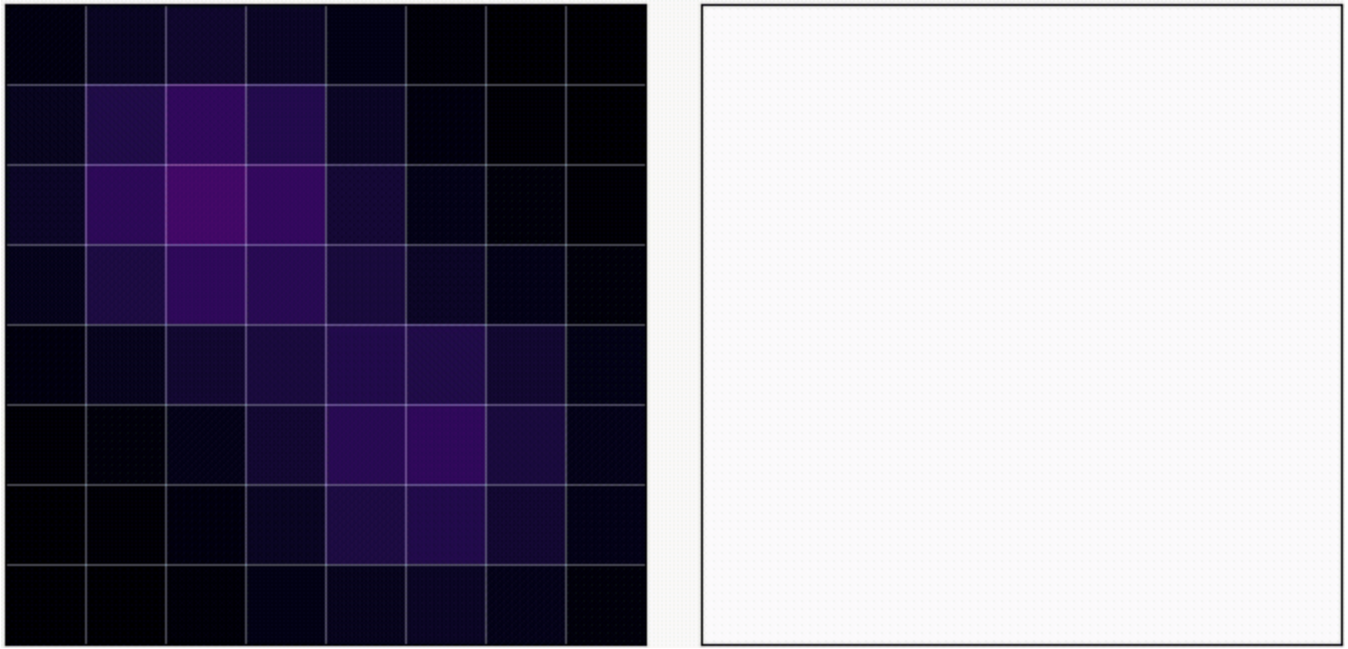
GaBP is a generalization of Jacobi

GaBP	Jacobi
$\mu_i^{(t+1)} = \frac{b_i - \sum_{j \in \mathcal{N}(i)} A_{ij} \mu_j^{(t)}}{A_{ii} + \sum_{j \in \mathcal{N}(i)} P_{j \to i}^{(t+1)}}$	$x_i^{(t+1)} = \frac{b_i - \sum_{j \in \mathcal{N}(i)} A_{ij} x_j^{(t)}}{A_{ii}}$
$v_i^{(t+1)} = \frac{1}{A_{ii} + \sum_{j \in \mathcal{N}(i)} P_{j \to i}^{(t+1)}}$	No variance update

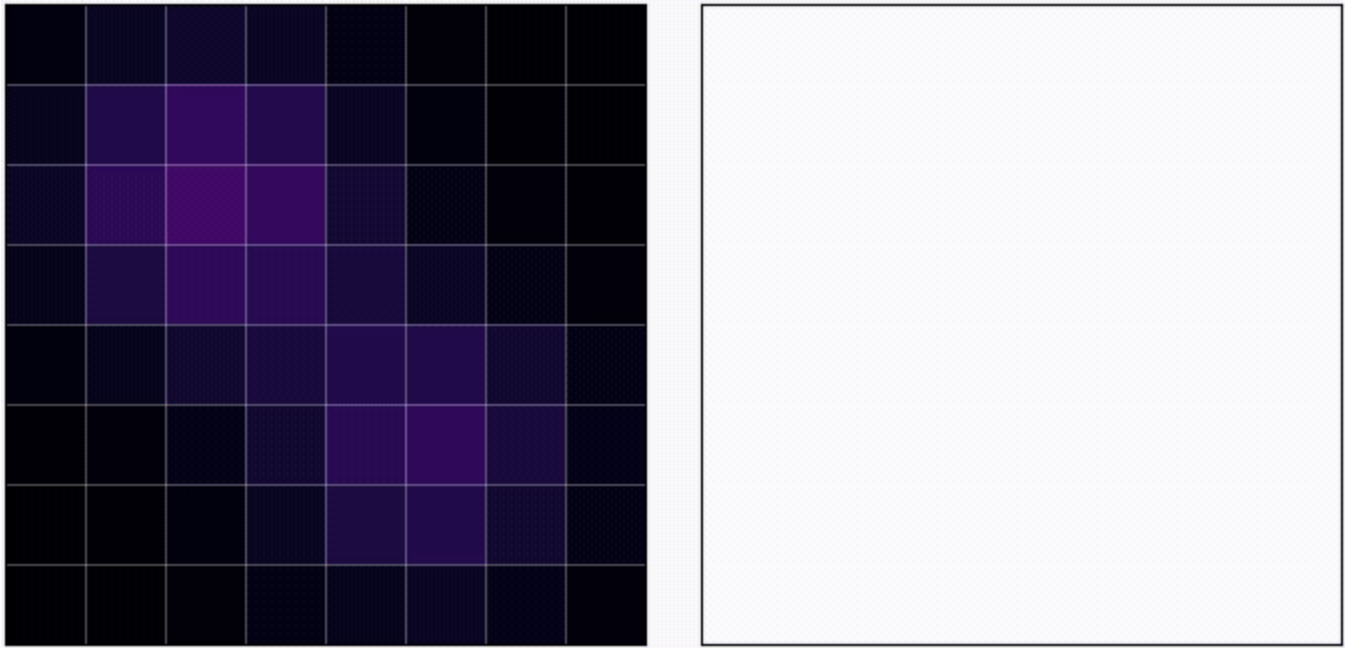
GaBP precision message: $P_{j \to i}^{(t+1)} = -\frac{A_{ij}^2}{A_{jj} + \sum_{k \in \mathcal{N}(j) \setminus \{i\}} P_{k \to j}^{(t)}}$

TU/e

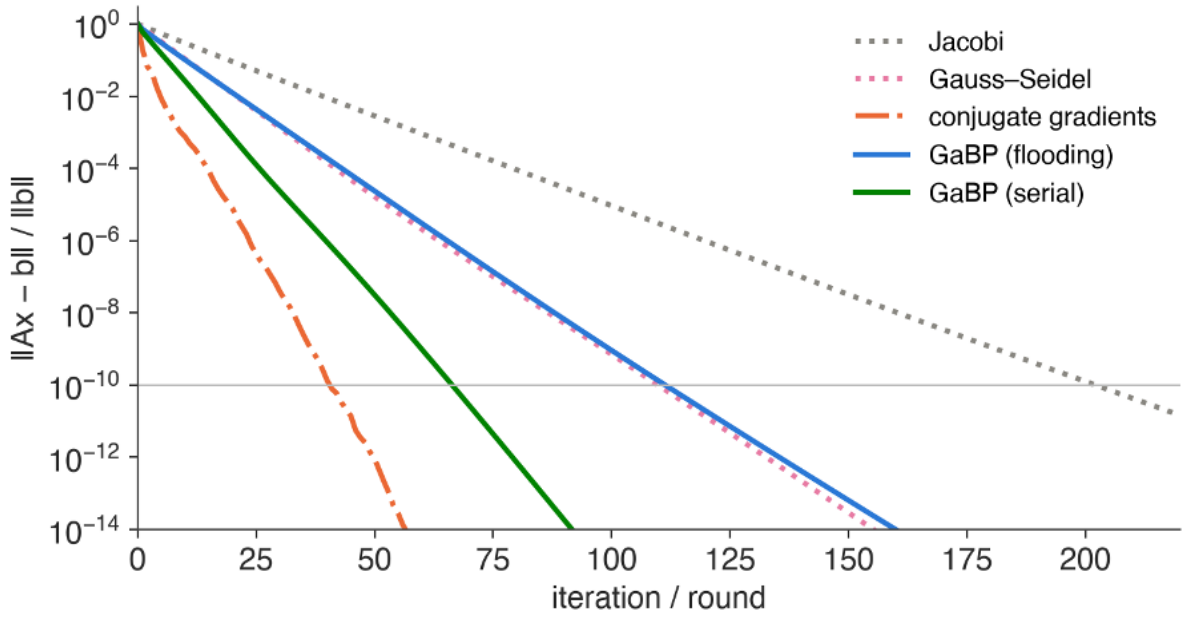
GaBP iterations



You can play with schedules!

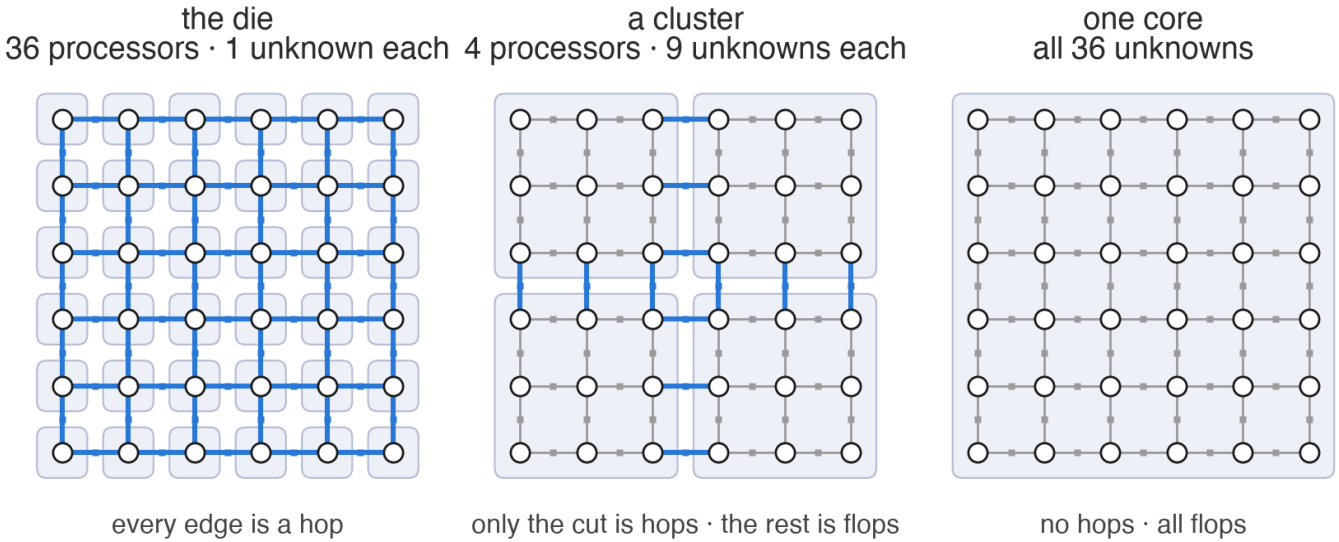


Is our generalization useful?



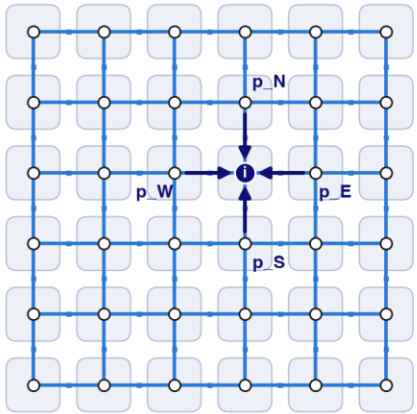
TU/e

GaBP is distributed algorithm



CG needs global synchronization steps

① the matrix–vector product A p_k — local



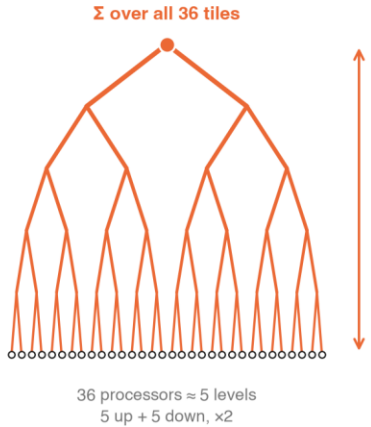
$(A\,p_k)_i = (4 + c)\cdot p_i - p\_N - p\_S - p\_E - p\_W$

four numbers from the neighbours · one hop

one CG iteration, tile by tile

$(A\,p_k)_i$	◀ four neighbours
$\alpha_k = r_k^T r_k \,/\, p_k^T A\,p_k$	sum over all tiles ▶
$x_{k+1} = x_k + \alpha_k\, p_k$	*local*
$r_{k+1} = r_k - \alpha_k\, A\, p_k$	*local*
$\beta_k = r_{k+1}^T r_{k+1} \,/\, r_k^T r_k$	sum over all tiles ▶
$p_{k+1} = r_{k+1} + \beta_k\, p_k$	*local*

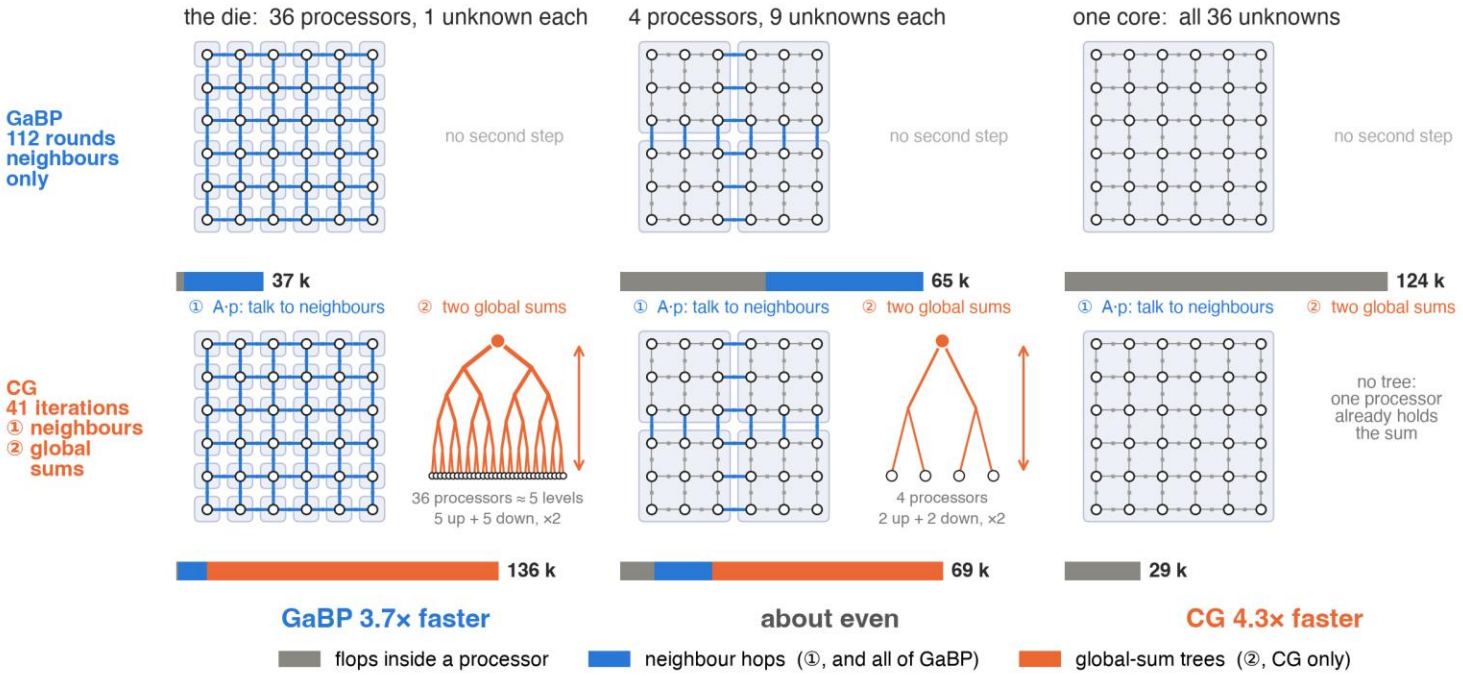
② the two scalars — two all-reduces



$p_k^T A\,p_k$ for α, then the new $r^T r$ for β — each a sum over every tile: partial sums go up, the total comes back down · $\log_2 P$ hops each way

TU/e

What algorithm wins and for what topology

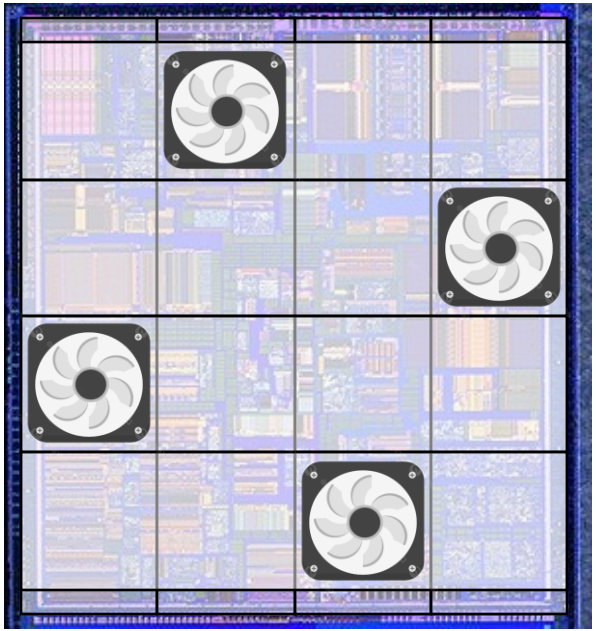


5. Stacking modular components

A Bayesian Jacobi solver

B Building

Adding some Fans



Before: one solve

$$A\,x = b$$

b measured by every tile; heat settles in microseconds

Now: one solve per time step

$$A\,x(t) = b(t)$$

the field is at steady state for the fans of the moment

$$b(t) = b(t-1) + w(t), \qquad w(t) \sim \mathcal{N}(0, \Lambda^{-1})$$

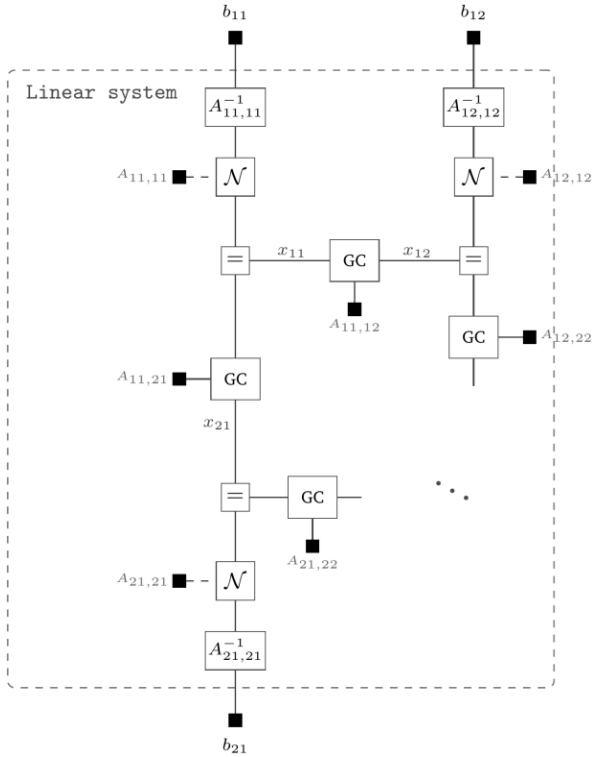
a fan has inertia: it drifts, it cannot jump

$$y(t) = b(t) + v(t), \qquad v(t) \sim \mathcal{N}(0, \tau^{-1} I)$$

a tachometer on each fan: noisy, sometimes dark

TU/e

Translating from the FFG into Code

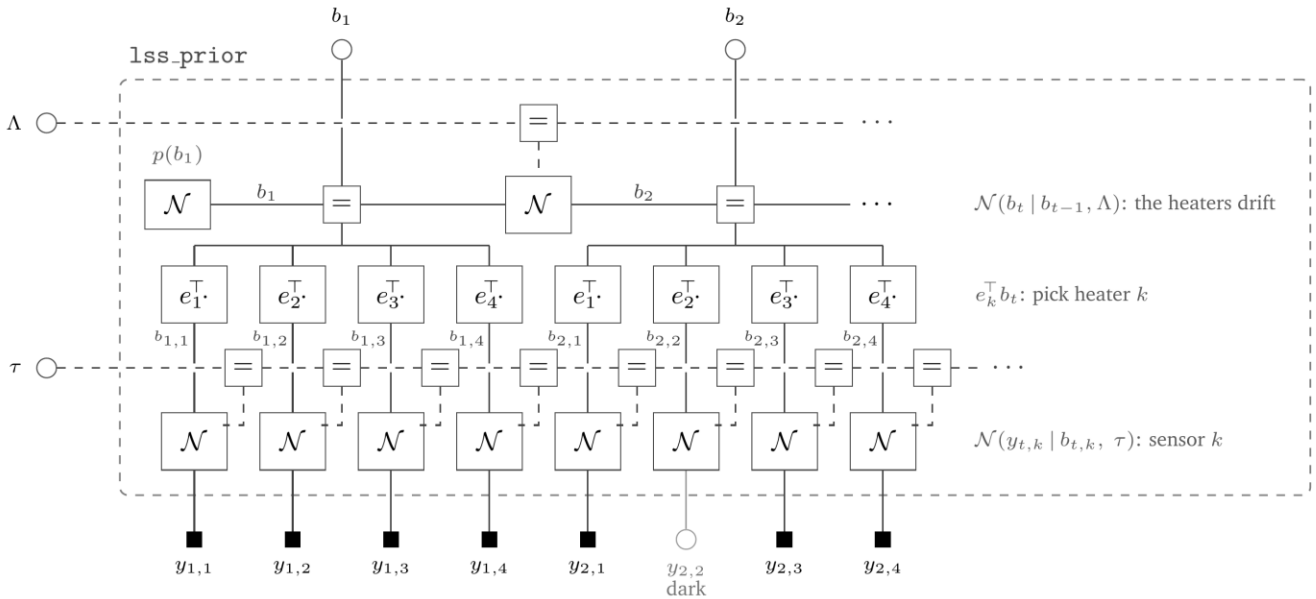


linear_system.jl

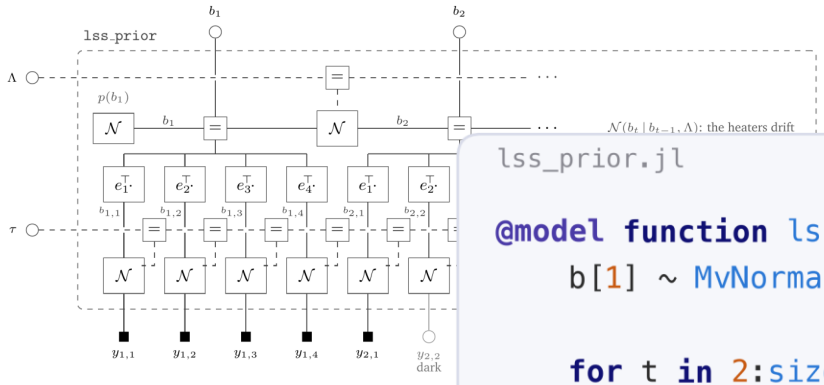
```julia
@model function linear_system_model(b, A)
    n = length(b)

    for i in 1:n                                # one self factor per cell
        x[i] ~ Normal(mean = b[i] / A[i, i],
                      precision = A[i, i])
    end

    for i in 1:(n - 1), j in (i + 1):n          # one coupling per non-zero
        if !iszero(A[i, j])
            x[j] ~ GaussianCoupling(x[i], -A[i, j])
        end
    end
end
```

TU/e

A linear state space model is a probabilistic block



A linear state space in code

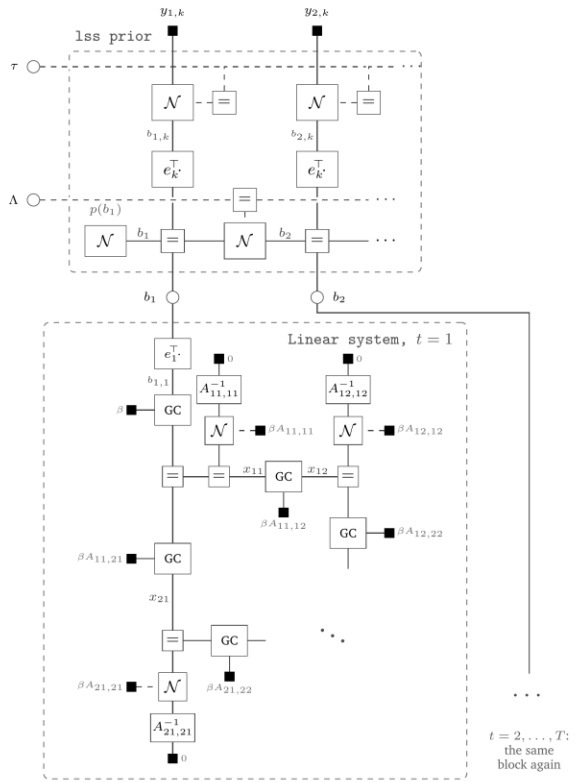


lss_prior.jl

```
@model function lss_prior(y_b, b, Λ, τ, dim_latent)
    b[1] ~ MvNormalMeanPrecision(zeros(dim_latent),
                                 0.01 * diageye(dim_latent))
    for t in 2:size(y_b, 1)
        b[t] ~ MvNormalMeanPrecision(b[t-1], Λ)    # drift
    end
    for t in 1:size(y_b, 1), k in 1:dim_latent
        y_b[t, k] ~ NormalMeanPrecision(           # sensor k
            dot(b[t], unit(k, dim_latent)), τ)
    end
end
```

TU/e

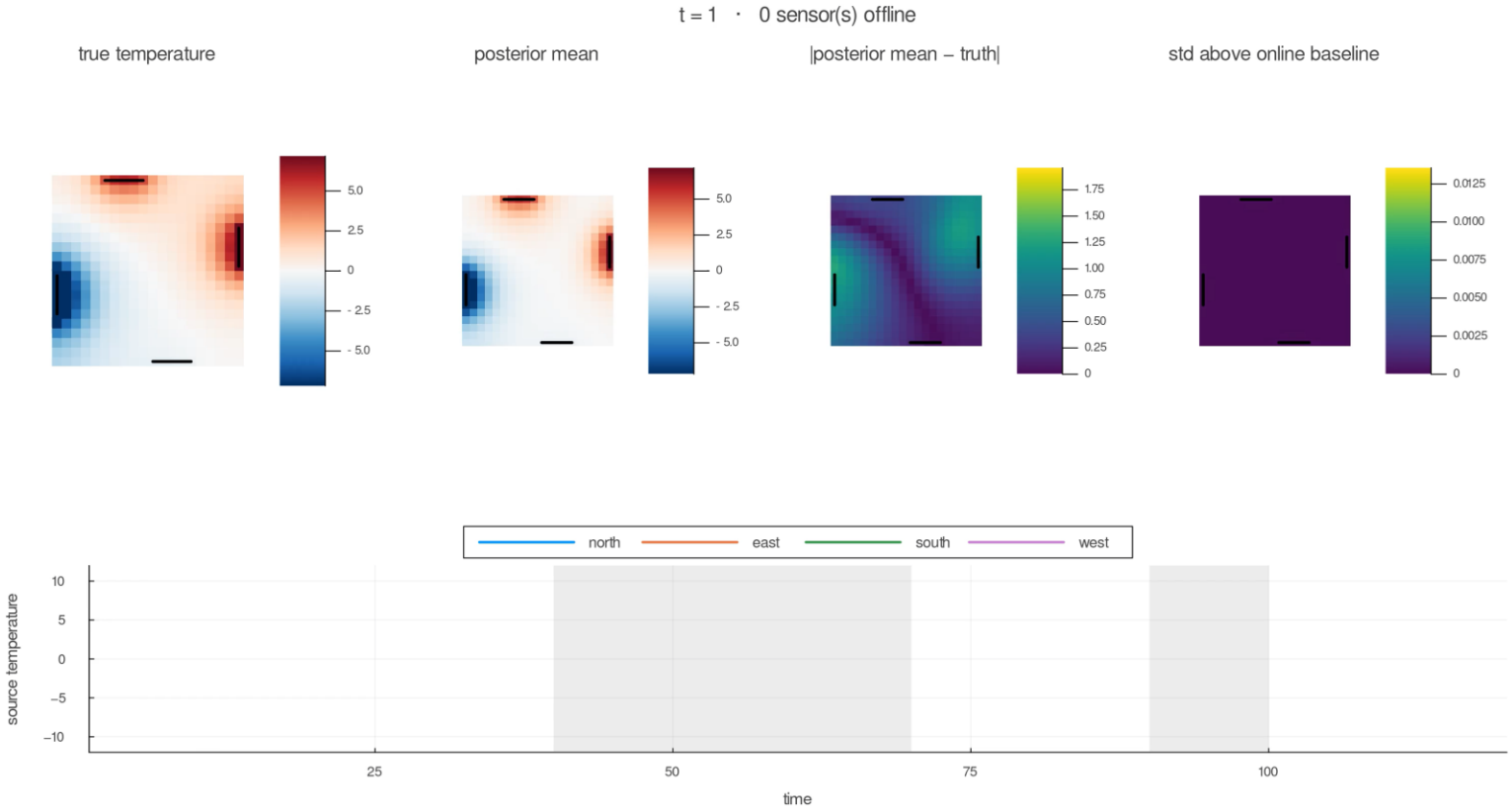
Compositional payoff



heat_grid_model.jl

```julia
@model function heat_grid_model(y_b, b0, A, β, Λ, τ, source_cells)
    T, n, K = size(y_b, 1), size(A, 1), length(source_cells)

    b ~ lss_prior(y_b = y_b, Λ = Λ, τ = τ, dim_latent = K)

    for t in 1:T
        for i in 1:n                                    # self factors
            x[t, i] ~ Normal(mean = b0[i] / A[i, i],
                             precision = β * A[i, i])
        end
        for k in 1:K, i in source_cells[k]              # heater → its cells
            x[t, i] ~ GaussianCoupling(dot(b[t], unit(k, K)), β)
        end
        for i in 1:n-1, j in i+1:n                      # cell ↔ cell, from A
            if !iszero(A[i, j])
                x[t, j] ~ GaussianCoupling(x[t, i], -β * A[i, j])
            end
        end
    end
end
```

Let's do some Infer!



Thanks for your attention!

Extra: convergence of Jacobi iteration

Jacobi updates are a fixed-point iteration with iteration matrix $M_J = D^{-1}(L + U)$.

It can be shown that it converges for every $x^{(0)}$ if and only if the spectral radius $\rho(M_J) < 1$.

Take, for example, a chain tiling of the die such that A is tridiagonal with values $(-1,2,-1)$.

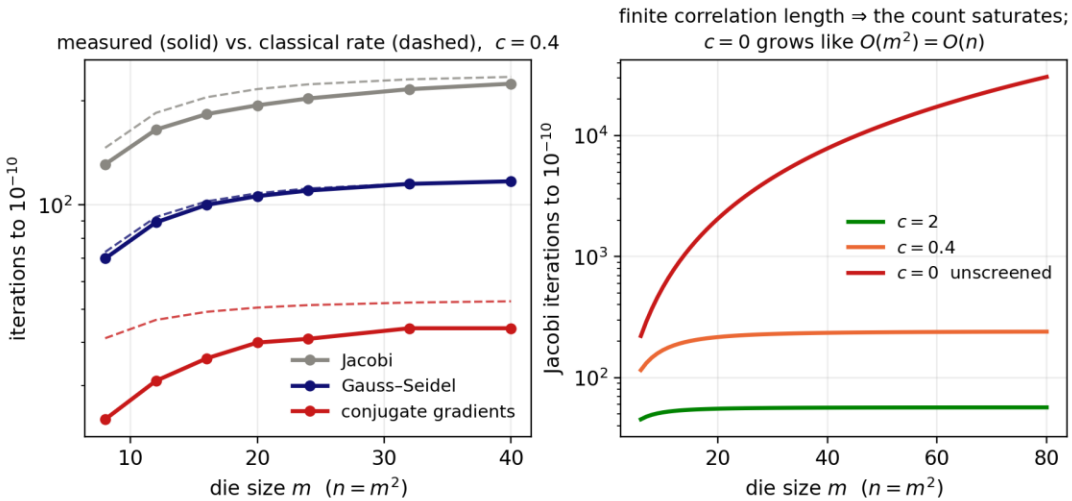
Then, the eigenvalues of M_J are $\lambda_j = \cos(\frac{j\pi}{n+1})$ for $j = 1, \dots, n$.

This means the spectral radius is $\cos(\frac{\pi}{n+1})$, which is roughly $1 - \frac{\pi^2}{2(n+1)^2}$ and less than 1.

Extra: error rates

For $\kappa = \lambda_{\max} / \lambda_{\min}$, the error rate of the conjugate gradient method is

$$\| x^{(t)} - x^* \|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^t \| x^0 - x^* \|_A$$



Extra: CG is posterior mean of BayesCG

If you choose $x^{(0)} = 0$ and $\Sigma^{(0)} = A^{-1}$, then the posterior mean is exactly the result of CG:

$$\begin{aligned} x^{(m)} &= x^{(0)} + \Sigma^{(0)} A^T S \left(S^T A \Sigma^0 A^T S\right)^{-1} S^T \left(b - A x^{(0)}\right) \\ &= 0 + A^{-1} A^T S \left(S^T A A^{-1} A^T S\right)^{-1} S^T (b - A0) \\ &= S \left(S^T A^T S\right)^{-1} S^T b \end{aligned}$$