Design and Implementation of Expectation Propagation Toolbox

Matthias Seeger
Probabilistic Machine Learning Laboratory
Ecole Polytechnique Fédérale de Lausanne
INR 112, Station 14, CH-1015 Lausanne
matthias.seeger@epfl.ch

February 4, 2014

1 Requirements

There is a need for an easy-to-use rapid prototyping toolbox for running expectation propagation (EP) approximate Bayesian inference on generalized linear models with non-Gaussian potentials. At least for a first instance, it is important not to be too ambitious, not to duplicate existing, well-implemented efforts, but also to support a range of EP variants of different kind (in particular, the fully factorized *and* the fully coupled case). A primary goal is to obtain a first working implementation rapidly, so concentrate on familiar cases and ignore difficult settings for now.

Requirements:

- Python/C++ implementation. First prototype could be done in Matlab, this could also be used as test code for the real thing
- Support generalized linear models with Gaussian and non-Gaussian potentials. Examples:
  - Logistic and probit regression. Robust regression. Quantile regression. Poisson regression
  - Multivariate probit regression
  - Sparsity priors. Sparse linear models
- Support *both* modes:
  - Fully coupled Gaussian backbone
  - Fully factorized Gaussian backbone
- Support sequential and parallel updating in coupled mode, sequential updating in factorized mode
- Support damping, skipping, other local update heuristics to avoid breakdown

- Coupled mode: Marginals maintained in moment parameterization. Factorized mode: Marginals, messages maintained in natural parameterization
- Fully generic interface for univariate potentials. If a potential is implemented, it can be used in all modes
- Generic support of coupling factors in coupled mode
- As generic as reasonable in factorized mode. Possibly restrict to sparse matrix coupling factors, which would be easiest to implement

The following features will not be supported, at least in the near future:

- Models with Gaussian factors, which are not in GLM form
  - Gaussian process models: Covariance matrix given. This case is already well supported by the GP-ML toolbox
  - Gaussian MRF potentials: Inverse covariance matrix given
- Non-Gaussian factors in backbone or messages:
  - Discrete variable inference
  - Variance hyperparameters (this could be supported in restricted cases in the future)
- Probabilistic programming interface
- General multivariate potentials
- Approximate Gaussian computations in the coupled case (conjugate gradients, sampling of Gaussian variances)
- Double loop algorithm steps as fallback to guarantee convergence

The following features will not be supported in the first implementation, but should at some point be considered:

- Generic support for univariate potentials, given only minimal information (log potential function, support, maybe discontinuities). Could first be implemented by calling adaptive quadrature functions from `quadpack`. To make this fast, consider automatic tabulation (with linear interpolation).
- Local projections other than moment matching: α-divergences, variational message passing. More general, we should support other variational relaxations, in particular the local variational method.
- Fractional (or power) EP
- Specific measures for dealing with negative π_j or π_{ji}. Could be as simple as forcing them to be ≥ 0. Also, [3] could be helpful
- Output of EP free energy (for monitoring)

- Integrate out certain hyperparameters (such as prior and noise variances) within EP, using non-Gaussian exponential families. Identify relevant use cases (noise variance, ARD) and implement a solution which works for these. This is a much better way of dealing with hyperparameters than optimizing them. Limit this to "simple cases" (stay clear of `Infer.NET` hard cases.
- Hyperparameter learning by minimizing EP free energy
- Representation for coupled Gaussian if there are more variables than factors (Woodbury)
- Coupled sequential updating mode: Optimized scheduling of EP updates (see Section 3.4). In fact, the coupled sequential mode with random ordering is too slow to be useful
- Mixed mode: Coupled Gaussian on blocks of variables (blockdiagonal factorization)
- EP updates where cavity marginal is undefined. This makes sense for specific potentials only (e.g., spike&slab)
- Backtracking to previous iterations for recovery (??)

Other projects with similar goals are:

- `BayesPy`: Python framework. Setup more like probabilistic programming. Implements variational message passing (factorized mode) only. Single developer (Aalto). Unlikely to contain EP in the near future, certainly not coupled mode
- `Infer.NET`: Most sophisticated and ambitious framework. Probabilistic programming, general variables, scalable. Supported (now) by full team of developers. Difficult to use for non-Windows people (.NET, C#). Closed source, hard to modify (requires large initial effort)
- `BayesBlocks`: C++/Python, implements particular "Bayes Blocks" framework, probably based on variational mean field with conjugate families, or the variational Gaussian approximation. Several developers (Aalto). Does not contain EP, restricted to what they are doing.
  Note: Could be good source of inspiration for Python implementation
- `STAN`, `BUGS`: This is MCMC. In particular, STAN could be really interesting to get MCMC up and running quickly on a fairly general model

1.1 Initial Project Plan

Here is what has to be done (this could shift a lot):

- Work out the maths, for both coupled and factorized mode
- Isolate common blocks, start to plan interfaces

- Agree on initial schema for algorithm in either mode, in particular which "convergence measures" are supported in the first version
- Define 2–3 use cases, which will be supported in example code
- Design:
  - Services. Interfaces. Be restrictive for first version!
  - Classes
  - How to implement this in Matlab? How to implement this in Python? Rough sketch in either case.
    For Python: What do I need to learn about to get this done? Who can I ask?
- Implement Matlab prototype:
  - Study Hannes `glm-ie`. Can I use code from there? Get in touch
  - Implement this rapidly, no code optimization. Minimize the use of MEX files (only for EP updates)
  - Implement test code for 1–2 use cases. Ask Hannes how one could compare things
- Python/C++ implementation:
  - Find suitable Python packages for the primitives
  - Interface to C++ code is best done in Cython, which offers great support for NumPy extensions
  - Implement a demo with visualizations of EP updates, thereby getting used to MatPlotLib as well

1.2 Ideas for Useful Directions

Once the toolbox exists, how can we make it more generally useful for relevant problems? Here, we collect ideas for directions.

- Safe convergent variant in the factorized case. Since we limit ourselves to bipartite graphs and Gaussian backbones, this could be achievable. First, have to really understand the variational problem in the factorized case (e.g., by mapping it to the EC formulation). Then, consider some unrolled version of a double loop algorithm, which runs message passing most of the time.
  Note: This is a very important point. One of the major drawbacks of factorized EP for large scale applications, compared to convex optimization methods, is the lack of a convergence proof.
- Parallelization in MapReduce fashion. In coupled mode, the number of message parameters is minimal (just $O(m)$), but the posterior representation is very large. In factorized mode, we need $O(\mathrm{nnz}(\boldsymbol{B}))$ message parameters, which can be maintained decentrally, and the representation is minimally small (just $O(n)$). Key questions: Can the messages be compressed somehow in factorized mode? Is there something between coupled and factorized which allows for a better trade-off?

2 Setup. Notation

We restrict ourselves to continuous variables and Gaussian backbones. The model is that of a bipartite factor graph with linear couplings. The posterior distribution to be approximated is

$$p(\boldsymbol{x}) = Z^{-1} \prod_{j=1}^{m} t_j(s_j), \quad \boldsymbol{s} = \boldsymbol{B}\boldsymbol{x} \in \mathbb{R}^m, \quad \boldsymbol{x} \in \mathbb{R}^n.$$

This setup includes potentials on the x_i, since part of $\boldsymbol{B}$ can be the identity. The potentials are univariate non-negative. They can be Gaussian, in fact this could be the default. Any offsets must be integrated into the potentials.

In general, we a *Gaussian backbone distribution* $q(\boldsymbol{x})$. This means that $q(\boldsymbol{x})$ represents $p(\boldsymbol{x})$, even though the best approximation to the marginal $p(s_j)$ is not necessarily given by $q(s_j)$. $q(\boldsymbol{x})$ is parameterized in terms of EP parameters (also term approximations, or factor-to-variable messages). In the *training* stage, these are adjusted by running the EP algorithm (different modes).

Once converged, *predictions* on a test set can be performed. Given a factor $\boldsymbol{B}_*$, the most basic prediction is given by Gaussian marginals $q(s_{*j})$, where $\boldsymbol{s}_* = \boldsymbol{B}_*\boldsymbol{x}$. Optionally, potentials $t_{*j}(s_{*j})$ can be passed. In this case,

$$\hat{p}_{*j}(s_{*j}) = Z_{*j}^{-1} t_{*j}(s_{*j}) q(s_{*j})$$

is the predictive distribution, and $\log Z_{*j}$, means and variances can be returned.

HIER: MORE?

2.1 Notation

In general, i is an index of variables x_i, j an index over factors $t_j(s_j)$. $\sum_i$ is short for $\sum_{i=1}^{n}$, $\sum_j$ short for $\sum_{j=1}^{m}$, the same for products.

All marginals and term approximations (messages) are Gaussian, typically univariate. Gaussians on x_i have moment parameters (mean, variance) denoted by μ, σ^2, for example $q(x_i) = N(\mu_i, \sigma_i^2)$. The natural parameterization is given by

$$q(x_i) \propto N^U(\beta_i, \pi_i) := e^{\beta_i x_i - \frac{1}{2}\pi_i x_i^2}.$$

The conversion is

$$\sigma_i^2 = \frac{1}{\pi_i}, \quad \mu_i = \frac{\beta_i}{\pi_i}.$$

For proper distributions, $\pi_i > 0$ (typically even $\pi_i \geq \varepsilon > 0$). For messages, single π may be negative. We also need marginals on $\boldsymbol{s} = \boldsymbol{B}\boldsymbol{x}$. Their moment parameters are denoted by h, ρ, for example $q(s_j) = N(h_j, \rho_j)$.

HIER: MORE?

3 Coupled Gaussian Backbone

In coupled mode, $q(\boldsymbol{x})$ is a general Gaussian. We only require $O(m)$ EP parameters, but the updates require dense Gaussian computations. The players are:

- Marginals $q(s_j) = N(h_j, \rho_j)$, where $\boldsymbol{s} = \boldsymbol{B}\boldsymbol{x}$
- Term approximations $m_j(s_j) = N^U(\beta_j, \pi_j)$.
  It is OK for single π_j to be negative, as long as cavity marginals and marginals are well defined
- Cavity marginals $q_{-j}(s_j) = N(h_{-j}, \rho_{-j})$.
  Require these to be defined, and ρ_{-j} to not be too large. In particular, we try to bound ρ_{-j}/ρ_j

The model may contain Gaussian potentials $t_j(s_j)$. These are directly replaced by fixed $m_j(s_j)$, their parameters are never updated.

We have that

$$q(\boldsymbol{x}) \propto \prod_j m_j(s_j) \propto \exp\left(\boldsymbol{\beta}^T \boldsymbol{B}\boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^T \boldsymbol{B}^T \boldsymbol{\Pi} \boldsymbol{B}\boldsymbol{x}\right).$$

The representation mainly consists of the Cholesky factor $\boldsymbol{L}$ and the vector $\boldsymbol{c}$:

$$\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{B}^T \boldsymbol{\Pi} \boldsymbol{B}, \quad \boldsymbol{c} = \boldsymbol{L}^{-1}\boldsymbol{B}^T\boldsymbol{\beta}.$$

With this:

$$\mathrm{E}_q[\boldsymbol{s}] = \boldsymbol{B}(\boldsymbol{L}\boldsymbol{L}^T)^{-1}\boldsymbol{B}^T\boldsymbol{\beta} = \boldsymbol{B}\boldsymbol{L}^{-T}\boldsymbol{c}, \quad \mathrm{Cov}_q[\boldsymbol{s}] = \boldsymbol{B}(\boldsymbol{L}\boldsymbol{L}^T)^{-1}\boldsymbol{B}^T.$$

There are different submodes:

- Sequential updating, marginals on demand
- Sequential updating, all marginals up-to-date. This is useful to support optimized update scheduling
- Parallel updating

In the latter two, we also represent

$$\boldsymbol{h} = [h_j],\ h_j = \mathrm{E}_q[s_j], \quad \boldsymbol{\rho} = [\rho_j],\ \rho_j = \mathrm{Var}_q[s_j].$$

In any of these, the EP algorithm iterates between EP updates on potentials $t_j(s_j)$ and updating the representation.

3.1 Updating the Representation

Suppose that

$$\pi_j' = \pi_j + \Delta\pi_j, \quad \beta_j' = \beta_j + \Delta\beta_j.$$

First we deal with sequential updating. For marginals on demand, we compute

$$\boldsymbol{v}_j = \boldsymbol{L}^{-1}\boldsymbol{b}_j, \quad \boldsymbol{b}_j = \boldsymbol{B}^T\boldsymbol{\delta}_j,$$

then

$$h_j = \boldsymbol{\delta}_j^T\boldsymbol{B}\boldsymbol{L}^{-T}\boldsymbol{c} = \boldsymbol{v}_j^T\boldsymbol{c}, \quad \rho_j = \|\boldsymbol{v}_j\|^2.$$

This is done before the EP update, and $\boldsymbol{v}_j$ is retained. Afterwards,

$$\boldsymbol{L}'(\boldsymbol{L}')^T = \boldsymbol{L}\boldsymbol{L}^T + \Delta\pi_j\boldsymbol{b}_j\boldsymbol{b}_j^T, \quad \boldsymbol{L}'\boldsymbol{c}' = \boldsymbol{L}\boldsymbol{c} + \Delta\beta_j\boldsymbol{b}_j.$$

$(\boldsymbol{L}', \boldsymbol{c}')$ is computed from $(\boldsymbol{L}, \boldsymbol{c})$ by Cholesky updating ($\Delta\pi_j > 0$) or downdating ($\Delta\pi_j < 0$) [5]. Here, if $|\Delta\pi_j|$ is too small, the update is skipped altogether. These routines may also require $\boldsymbol{v}_j$.
Note: Numerically, a Cholesky update cannot fail, but a downdate can. This will happen iff $1 + (\Delta\pi_j)\rho_j \approx 0$ or even negative. In this case, damping should be applied until $|\Delta\pi_j|$ is small enough. Cost:

- Cholesky update/downdate: $O(n^2)$
- One backsubstitution with $\boldsymbol{L}$: $O(n^2)$
- Extract row from $\boldsymbol{B}$

Sequential updating, all marginals up-to-date. The Woodbury formula gives:

$$(\boldsymbol{L}'(\boldsymbol{L}')^T)^{-1} = (\boldsymbol{L}\boldsymbol{L}^T)^{-1} - f_j\boldsymbol{L}^{-T}\boldsymbol{v}_j(\boldsymbol{L}^{-T}\boldsymbol{v}_j)^T, \quad f_j = \frac{\Delta\pi_j}{1 + \Delta\pi_j\rho_j}.$$

Some algebra:

$$\boldsymbol{h}' = \boldsymbol{h} + e_j\boldsymbol{w}_j, \quad \boldsymbol{w}_j = \boldsymbol{B}\boldsymbol{L}^{-T}\boldsymbol{v}_j, \quad e_j = \frac{\Delta\beta_j - \Delta\pi_j h_j}{1 + \Delta\pi_j\rho_j}.$$

And

$$\boldsymbol{\rho}' = \boldsymbol{\rho} - f_j\boldsymbol{w}_j^2.$$

Cost:

- Cholesky update/downdate: $O(n^2)$
- Backsubstitution with $\boldsymbol{L}$, $\boldsymbol{L}^T$: $O(n^2)$
- Extract row from $\boldsymbol{B}$. Matrix-vector multiplication with $\boldsymbol{B}$

This is slightly more than for marginals on demand, but the marginals can be used to save on EP updates.

For parallel updating, the representation and marginals are recomputed from scratch. This should also be done in sequential mode: refreshing after each sweep over all potentials. We compute $\boldsymbol{h}$ by backsubstitutions and $\boldsymbol{B}$-MVM. Then, the inverse $(\boldsymbol{L}\boldsymbol{L}^T)^{-1}$ is computed from $\boldsymbol{L}$. Finally, $\boldsymbol{\rho}$ is computed by calling the primitive

$$\mathrm{diag}^{-1}\left(\boldsymbol{B}\boldsymbol{M}\boldsymbol{B}^T\right).$$

A default implementation would compute this as

$$\sum_i (\boldsymbol{B}\boldsymbol{M}_{\cdot,i}) \circ \boldsymbol{B}_{\cdot,i}.$$

Cost:

- Primitive: $\boldsymbol{B}(\mathrm{diag}\,\boldsymbol{\pi})\boldsymbol{B}^T$
- Cholesky decomposition
- If marginals: Inverse from Cholesky factor, and $\mathrm{diag}^{-1}(\boldsymbol{B}\boldsymbol{M}\boldsymbol{B}^T)$ primitive

3.2 EP Update

The input to an EP update at $t_j(s_j)$ is given by h_j, ρ_j, β_j, π_j. Define $q_{-j}(s_j) \propto q(s_j)/m_j(s_j)$ and $\hat{p}_j(s_j) = Z_j^{-1} t_j(s_j) q_{-j}(s_j)$. The output of the EP update is $m'_j(s_j) = N^U(\beta'_j, \pi'_j)$ s.t. $q'(s_j) \propto q_{-j}(s_j) m'_j(s_j)$ and $\hat{p}_j(s_j)$ have the same mean and variance. This is the full EP update, damping is discussed below. In our initial implementation, we require that the cavity marginals $q_{-j}(s_j)$ are proper Gaussians, even though in principle EP can be run with only the $\hat{p}_j(s_j)$ being properly defined (which makes sense for spike&slab potentials, for example).

If $q_{-j}(s_j) = N(h_{-j}, \rho_{-j})$, then

$$\rho_{-j} = \frac{\rho_j}{1 - \pi_j \rho_j}, \quad h_{-j} = \frac{h_j - \beta_j \rho_j}{1 - \pi_j \rho_j}.$$

This computation can fail. We require that $1 - \pi_j \rho_j \geq \varepsilon$, which limits $\rho_{-j}/\rho_j \leq 1/\varepsilon$. If this is not the case, the update could be skipped. We cannot just lower π_j. It is probably best to avoid or modify EP updates which would result in undefined cavity marginals. A mechanism is described at the end of this section.

Denote mean and variance of $\hat{p}_j(s_j)$ by $\hat{h}_j$, $\hat{\rho}_j$. If

$$\alpha_j := \frac{\partial \log Z_j}{\partial h_{-j}}, \quad \nu_j := -\frac{\partial \log Z_j}{\partial h_{-j}^2},$$

then we have

$$\hat{h}_j = h_{-j} + \alpha_j \rho_{-j}, \quad \hat{\rho}_j = \rho_{-j}(1 - \nu_j \rho_{-j}).$$

It turns out that (α_j, ν_j) are sufficient outputs for t_j-specific EP update code, where inputs should be (h_{-j}, ρ_{-j}) or $(h_j, \rho_j, \beta_j, \pi_j)$. The latter would be more general: we can always pass the cavity marginals and 0 for the EP parameters. The new EP parameters are

$$\pi'_j = \frac{\nu_j}{1 - \nu_j \rho_{-j}}, \quad \beta'_j = \frac{h_{-j}\nu_j + \alpha_j}{1 - \nu_j \rho_{-j}}. \tag{1}$$

For log-concave potentials, $\nu_j > 0$. But π'_j can become negative for non-log-concave potentials. This is not necessarily a problem, but damping should be used in order to make sure that the Cholesky downdate can be performed, and that subsequent cavity marginals are defined.

Damping. Pick a damping factor $\eta \in [0, 1)$. If $\tilde{q}_j(s_j) = N(\hat{h}_j, \hat{\rho}_j)$, we update β', π' so that the new marginal is

$$q'(s_j) \propto q(s_j)^\eta \tilde{q}_j(s_j)^{1-\eta}.$$

The undamped EP update is obtained for $\eta = 0$. The larger η, the more damping. Skipping the update corresponds to $\eta = 1$. Damping is implemented by first computing the undamped updates $\tilde{\pi}_j$, $\tilde{\beta}_j$ as in (1), then

$$\pi'_j = \eta\pi_j + (1 - \eta)\tilde{\pi}_j, \quad \beta'_j = \eta\beta_j + (1 - \eta)\tilde{\beta}_j.$$

The damping factor η can also be chosen differently for each potential.

Avoid undefined cavity marginals. Suppose that marginal moments $\boldsymbol{h}$, $\boldsymbol{\rho}$ are part of the representation (parallel mode, or sequential with marginals up-to-date). We can use selective damping in order to ensure that cavity marginals can be computed. For the sequential mode, recall that $\boldsymbol{\rho}' = \boldsymbol{\rho} - f_j \boldsymbol{w}_j^2$ and $\boldsymbol{\pi}' = \boldsymbol{\pi} + (\Delta\pi_j)\boldsymbol{\delta}_j$. We determine $\min\{1 - \pi'_k \rho'_k\}$. If $< \varepsilon$, we do more damping. In this case, we can determine the smallest damping factor for which $\min\{1 - \pi'_k \rho'_k\} \geq \varepsilon$. For $k \neq j$:

$$1 - \pi'_k \rho'_k = 1 - \pi_k \rho_k + \frac{(\Delta\pi_j)\pi_k w_k^2}{1 + (\Delta\pi_j)\rho_j}.$$

If $\Delta\pi_j > 0$, everything is fine. Otherwise, we can set this equal to ε and solve for $\Delta\pi_j$, which then determines the minimum damping factor. For $k = j$:

$$1 - \pi'_j \rho'_j = 1 - (\pi_j + \Delta\pi_j)(\rho_j - f_j w_j^2).$$

One problem with setting the damping factor this way is that if $1 - \pi_k \rho_k$ becomes equal to ε, any further update with $\Delta\pi_j < 0$ is rejected, unless $w_k^2 = 0$.

In parallel mode, we check whether $\min\{1 - \pi'_k \rho'_k\} \geq \varepsilon$. If not, we apply damping. The simplest approach is to use some increasing schedule for η and to recompute $\boldsymbol{\rho}'$, until the constraints are met. This can be quite expensive. If $k_* = \operatorname{argmin}\{1 - \pi'_k \rho'_k\}$, we could first determine the damping constant for which $1 - \pi'_{k_*} \rho'_{k_*} \geq 1.2\varepsilon$ (or so), then to try this value.

3.3 Interface for Coupling Factor $\boldsymbol{B}$

The following services are required for $\boldsymbol{B}$:

- Row/column: $\boldsymbol{B}_{\cdot,i}$, $\boldsymbol{B}_{j,\cdot}^T$

- Matrix-vector multiplication: $\boldsymbol{B}\boldsymbol{v}$, $\boldsymbol{B}^T\boldsymbol{v}$
- $\boldsymbol{\pi} \mapsto \boldsymbol{B}^T(\operatorname{diag}\boldsymbol{\pi})\boldsymbol{B}$, where $\pi_j \geq 0$
- $\boldsymbol{M} \mapsto \operatorname{diag}^{-1}(\boldsymbol{B}\boldsymbol{M}\boldsymbol{B}^T)$, where $\boldsymbol{M}$ is a symmetric matrix

3.4 Active Scheduling of EP Updates

A major reason for keeping all marginals up-to-date is that EP updates can be scheduled actively. This technique is detailed in [7] and can lead to large savings, in particular when the model is changed in a minor way (e.g., some new data is appended) and we warmstart from the previous EP fixed point. It is based on the observation that an update of the representation is much more expensive than computing a large number of local EP updates, measuring the effect on the local marginal.

Let $q(s_j)$ and $q'(s_j)$ be the Gaussian marginal before and after a local EP update on the j-th potential. If only this update is done, the effect on the posterior can be measured by

$$\mathrm{D}[q'(\boldsymbol{x}) \,\|\, q(\boldsymbol{x})] = \mathrm{D}[q'(s_j) \,\|\, q(s_j)] =: \Delta_j.$$

If we keep all marginals up-to-date, each Δ_j is computed in $O(1)$. We maintain $\{\Delta_j\}$ for a large subset $J \subset \{1, \dots, q\}$. In each round, we compute Δ_j for all $j \in J$, then update on the score maximizer. We retain a fraction ω of the top-scorers in J and replace the bottom $1 - \omega$ fraction by new entries randomly drawn. Maintaining J is done only to balance the time for scoring with that for a representation update. If the latter dominates, we can work with $J = \{1, \dots, q\}$. Finally,

$$\Delta_j = \frac{1}{2}\left(\log(\rho_j/\rho'_j) + (\rho_j/\rho'_j)^{-1} - 1 + (h'_j - h_j)^2/\rho_j\right).$$

Since $q'(s_j) \propto q(s_j) e^{(\Delta\beta_j)s_j - \frac{1}{2}(\Delta\pi_j)s_j^2}$, then

$$\rho'_j = \frac{\rho_j}{1 + \Delta\pi_j\rho_j}, \quad h'_j = \frac{h_j + \Delta\beta_j\rho_j}{1 + \Delta\pi_j\rho_j}.$$

Plugging this in:

$$\Delta_j = \frac{1}{2}\left(\log\kappa_j + \frac{1}{\kappa_j}\left(1 + \rho'_j(\Delta\beta_j - \Delta\pi_j\rho_j)^2\right) - 1\right), \quad \kappa_j := 1 + \Delta\pi_j\rho_j.$$

Note: We could also use Δ_j as alternative statistic in order to decide upon EP convergence.

4 Factorized Gaussian Backbone

For the factorized mode, we assume that $\boldsymbol{B}$ is a sparse matrix with rows $\boldsymbol{b}_j$. Denote the index of non-zero entries in $\boldsymbol{b}_j$ by V_j. When writing $\sum_i$ in the context of a sequential update on $t_j(s_j)$, we mean $\sum_{i \in V_j}$. Implementation details are discussed further below.

In the factorized mode, it makes sense to represent marginals and messages in natural parameters. The representation consists of marginals $q(x_i)$ and term approximations $m_{ji}(x_i)$.

Marginals are $q(x_i) \propto N^U(\beta_i, \pi_i)$. Term approximations are $m_{ji}(x_i) = N^U(\beta_{ji}, \pi_{ji})$. These can be interpreted as factor-to-variable messages $t_j \to x_i$. We always have that

$$\sum_j \beta_{ji} = \beta_i, \quad \sum_j \pi_{ji} = \pi_i.$$

It is OK that $\pi_{ji} < 0$ for single messages, but all cavity marginals have to be well defined ($\pi_{-ji} \geq \varepsilon$) as well as all marginals ($\pi_i \geq \varepsilon$).

4.1 Sequential Updating

We first discuss sequential updating (parallel updating may not be useful, due to problems with convergence). Suppose we update on $t_j(s_j)$. The cavity marginals are $q_{-j}(x_i) \propto N^U(\beta_{-ji}, \pi_{-ji})$, where

$$\beta_{-ji} = \beta_i - \beta_{ji}, \quad \pi_{-ji} = \pi_i - \pi_{ji}, \quad i \in V_j.$$

We require that $\pi_{-ji} \geq \varepsilon > 0$ for all $i \in V_j$, this should be ensured permanently (see below). For the EP update, we determine $q_{-j}(s_j)$ and $\hat{p}_j(s_j) \propto t_j(s_j) q_{-j}(s_j)$. The new $q'(\boldsymbol{x})$ is the factorizing Gaussian with same means and variances as $\hat{p}_j(\boldsymbol{x}) \propto t_j(s_j) \prod_i q_{-j}(x_i)$.

First, $q_{-j}(s_j) = N(h_{-j}, \rho_{-j})$, where

$$h_{-j} = \sum_i b_{ji} \beta_{-ji} / \pi_{-ji}, \quad \rho_{-j} = \sum_i b_{ji}^2 / \pi_{-ji}.$$

Now, we do an EP update for $\hat{p}_j(s_j) = Z_j^{-1} t_j(s_j) q_{-j}(s_j)$, giving rise to α_j, ν_j (see Section 3.2). Define $\alpha_{ji} = \alpha_j b_{ji}$, $\nu_{ji} = \nu_j b_{ji}^2$. We show below that

$$\pi'_{ji} = \frac{\nu_{ji} \pi_{-ji}}{\pi_{-ji} - \nu_{ji}} = e_{ji} \nu_j \pi_{-ji}, \quad e_{ji} = \frac{1}{\pi_{-ji}/b_{ji}^2 - \nu_j}.$$

The last form can be used only if b_{ji}^2 is not too small. Also,

$$\beta'_{ji} = \frac{\beta_{-ji} \nu_{ji} + \pi_{-ji} \alpha_{ji}}{\pi_{-ji} - \nu_{ji}} = e_{ji} \left(\beta_{-ji} \nu_j + \pi_{-ji} \frac{\alpha_j}{b_{ji}} \right),$$

the last form only if b_{ji}^2 is not too small. For log-concave potentials, $\nu_j > 0$, therefore $\nu_{ji} \geq 0$ and $\pi'_{ji} \geq 0$. But π'_{ji} can become negative for non-log-concave potentials. This is not necessarily a problem, but damping should be applied to make sure that subsequent cavity marginals are defined (see below). Finally, the marginals are updated:

$$\pi'_i = \pi_{-ji} + \pi'_{ji}, \quad \beta'_i = \beta_{-ji} + \beta'_{ji}.$$

If $\nu_j < 0$, we have to ensure that $\pi'_i \geq \varepsilon$ for all $i \in V_j$. Otherwise, selective damping has to be applied.

Damping. Selective Damping. This is done as in Section 3.2:

$$\pi'_{ji} = \eta \pi_{ji} + (1 - \eta) \tilde{\pi}_{ji}, \quad \beta'_{ji} = \eta \beta_{ji} + (1 - \eta) \tilde{\beta}_{ji}.$$

Damping can be applied in general if convergence problems are detected. In the first implementation, no specific heuristics will be implemented for that.

Damping can be applied selectively in order to ensure that $\pi_{-ki} \ge \varepsilon$ for all k and $i \in V_k$ after the update on $t_j(s_j)$. To this end, we maintain $\kappa_i := \max_k \pi_{ki}$ for every i. After an update, we check whether $\pi_i' - \kappa_i' \ge \varepsilon$ for all $i \in V_j$ (note that κ_i' can be different from κ_i for $i \in V_j$). If this does not hold, we increase the damping. It can be tricky to maintain the κ_i up to date. A simple solution would be to compute κ_i for $i \in V_j$ before the update, but this is not very efficient. For each i, we maintain a sorted buffer with the K largest π_{ki}, together with the k indices. An update on t_j produces new π_{ji} for $i \in V_j$. There are several cases in which the buffer can be updated or will not change. A critical case is if j is in the buffer, but the new π_{ji} is smaller than all other entries. In this case, the buffer size shrinks by 1, j is removed. Once the buffer size shrinks to zero, it has to be recomputed.

Suppose that κ_i is available, and note that $\kappa_i' = \max\{\kappa_i, \pi_{ji}'\}$. First, the case $\tilde{\pi}_{ji} \ge \pi_{ji}$ is uncritical. Namely, then $\pi_{ji}' \ge \pi_{ji}$, and $\pi_i' \ge \pi_i \ge \varepsilon$. If $\pi_{ji}' \le \kappa_i$, then

$$\pi_i' - \kappa_i' = \pi_i' - \kappa_i \ge \pi_i - \kappa_i \ge \varepsilon.$$

And if $\pi_{ji}' > \kappa_i$, then

$$\pi_i' - \kappa_i' = \pi_{-ji} \ge \pi_i - \kappa_i \ge \varepsilon.$$

Now, assume that $\tilde{\pi}_{ji} < \pi_{ji}$, and the choice π_{ji}' is not accepted. First,

$$\kappa_i \ge \pi_{ji} > \pi_{ji}' \quad \Rightarrow \quad \kappa_i' = \kappa_i.$$

If $\kappa_i \ge 0$, we have to determine η such that $\pi_{ji}' - \kappa_i' = \varepsilon$:

$$\pi_{ji}' - \kappa_i' = \pi_i - \kappa_i + (1-\eta)(\tilde{\pi}_{ji} - \pi_{ji}) = \varepsilon \quad \Rightarrow \quad 1 - \eta = \frac{\pi_i - \kappa_i - \varepsilon}{\pi_{ji} - \tilde{\pi}_{ji}}.$$

Note that if $\pi_i - \kappa_i = \varepsilon$ and $\tilde{\pi}_{ji} < \pi_{ji}$, then $\eta = 1$, and the update must be skipped. If it is the case that $\pi_i - \kappa_i = \varepsilon$ for many i, the requirement may end up being too conservative. Finally, if in this case, $\kappa_i < 0$ (very unlikely!), then $\pi_i' \ge 0$ requires that

$$1 - \eta = \frac{\pi_i - \varepsilon}{\pi_{ji} - \tilde{\pi}_{ji}}.$$

Some care has to be taken at the beginning. First, if there are Gaussian potentials sitting on single x_i, they are never updated on. In this case, $\kappa_i = \max_k \pi_{ki}$ should run only over potentials k which we update on. Second, for models with a factorizing non-Gaussian prior on $\boldsymbol{x}$, it makes sense to start with $\pi_{ji} = 0$ for all likelihood potentials and to skip updates on prior potentials in the first sweep. Our precondition would be violated then. To stay out of trouble, for each i we pick some likelihood potential j s.t. $i \in V_j$ and set $\pi_{ji} = \frac{4}{3}\varepsilon$ (assuming that the initial π_{ji} values for prior potentials are larger). For small ε, this should not make any difference.

Derivation of Update. First,

$$q_{-j}(x_i) = N(\mu_{-ji}, \sigma_{-ji}^2), \quad \mu_{-ji} = \frac{\beta_{-ji}}{\pi_{-ji}}, \quad \sigma_{-ji}^2 = \frac{1}{\pi_{-ji}},$$

and $q_{-j}(s_j) = N(h_{-j}, \rho_{-j})$, $q'(s_j) = N(\hat{h}_j, \hat{\rho}_j)$, where

$$\hat{h}_j = h_{-j} + \alpha_j \rho_{-j}, \quad \hat{\rho}_j = \rho_{-j}(1 - \nu_j \rho_{-j}).$$

We require $q'(x_i)$, $i \in V_j$. First, $\mathrm{Cov}_{-j}[x_i, s_j] = \sigma^2_{-ji} b_{ji}$, so that

$$\mathrm{E}_{-j}[x_i | s_j] = \mu_{-ji} + \sigma^2_{-ji} b_{ji} \rho^{-1}_{-j}(s_j - h_{-j}), \quad \mu'_i = \mu_{-ji} + \sigma^2_{-ji}\alpha_{ji}, \quad \alpha_{ji} = \alpha_j b_{ji}.$$

Next, $\mathrm{Var}_{-j}[x_i | s_j] = \sigma^2_{-ji} - \sigma^4_{-ji} b^2_{ji} \rho^{-1}_{-j}$, so that

$$(\sigma^2_i)' = \sigma^2_{-ji} - \sigma^4_{-ji} b^2_{ji} \left(\rho^{-1}_{-j} - \mathrm{Var}_{q'}[s_j/\rho_{-j}]\right) = \sigma^2_{-ji}\left(1 - \nu_{ji}\sigma^2_{-ji}\right), \quad \nu_{ji} = b^2_{ji}\rho^{-2}_{-j}(\rho_{-j} - \hat{\rho}_j) = \nu_j b^2_{ji}.$$

The new EP parameters are therefore

$$\pi'_{ji} = \frac{\nu_{ji}}{1 - \sigma^2_{-ji}\nu_{ji}}, \quad \beta'_{ji} = \frac{\mu_{-ji}\nu_{ji} + \alpha_{ji}}{1 - \sigma^2_{-ji}\nu_{ji}}.$$

4.2 Initialization

A good initialization of EP parameters depends on the model. For a model with a proper Gaussian prior, we can use "ADF initialization". EP parameters for non-Gaussian potentials are set to zero, those for Gaussian potentials to the values represented by the prior. We then do not update on the Gaussian potentials in the first sweep.

ADF initialization is straightforward for a factorizing Gaussian prior on $\boldsymbol{x}$. For such a prior potential j: $V_j = \{i\}$ and $\boldsymbol{b}_j = 1$. In this case, we can fix corresponding EP parameters to the values represented by the prior and refrain from updating on these potentials at all (namely, the EP parameters will not change, and we can only run into trouble with undefined cavity distributions). The inference routine should be informed about this case.

If there is a Gaussian prior which is coupled on $\boldsymbol{x}$, or a Gaussian likelihood, things are less clear (contrary to the coupled mode, where all Gaussian potentials can be ignored). We cannot set the corresponding EP parameters independently of the current cavity distributions. The following derivation may give some idea. Suppose that $t_j(s_j) = N(y|s_j, \sigma^2)$, and that $|V_j| > 1$. We can make the assumption that

$$\pi_{-ji} = \pi_{-j} b^2_{ji}, \quad \beta_{-ji} = \beta_{-j} b_{ji}$$

for some $\pi_{-j} > 0$, β_{-j}. Under this assumption:

$$\rho_{-j} = \sum_i b^2_{ji}/\pi_{-ji} = |V_j|/\pi_{-j}, \quad h_{-j} = \sum_i b_{ji}\beta_{-ji}/\pi_{-ji} = |V_j|\beta_{-j}/\pi_{-j}.$$

From Section 5.2:

$$\nu_j = \frac{\pi_{-j}}{|V_j| + \sigma^2 \pi_{-j}}, \quad \alpha_j = \nu_j \left(y - |V_j|\beta_{-j}/\pi_{-j}\right).$$

Then,

$$\pi'_{ji} = \frac{\nu_{ji}\pi_{-ji}}{\pi_{-ji} - \nu_{ji}} = \frac{\pi^2_{-j} b^4_{ji}}{\pi_{-j} b^2_{ji}(|V_j| + \sigma^2\pi_{-j}) - \pi_{-j} b^2_{ji}} = \frac{b^2_{ji}}{(|V_j| - 1)/\pi_{-j} + \sigma^2}$$

and

$$\beta'_{ji} = \frac{\beta_{-ji}\nu_{ji} + \pi_{-ji}\alpha_{ji}}{\pi_{-ji} - \nu_{ji}} = \frac{\pi_{-j} b_{ji}^3 \beta_{-j} + \pi_{-j}^2 b_{ji}^3 (y - |V_j|\beta_{-j}/\pi_{-j})}{\pi_{-j} b_{ji}^2 (|V_j| + \sigma^2 \pi_{-j}) - \pi_{-j} b_{ji}^2} = b_{ji} \frac{y - (|V_j| - 1)\beta_{-j}/\pi_{-j}}{(|V_j| - 1)/\pi_{-j} + \sigma^2}.$$

If $|V_j| > 1$, these still depend on π_{-j}, β_{-j}. It may make sense to treat these as initialization parameters (same for all j). $\beta_{-j} = 0$ would simplify things.

4.3 Interface for Coupling Factor $\boldsymbol{B}$

It is not obvious to me how to do the sequential update efficiently in Matlab or Python. $\boldsymbol{B}$ could be stored as sparse matrix, but unless parallel updates are done, it is not clear whether this makes much sense.

For each j, we would maintain V_j, $\boldsymbol{\beta}_j = [\beta_{ji}]$, $\boldsymbol{\pi}_j = [\pi_{ji}]$ (the vectors in $\mathbb{R}^{|V_j|}$). Begin by extracting $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}(V_j)$, $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi}(V_j)$, then $\boldsymbol{\beta}_{-j} \leftarrow \boldsymbol{\beta} - \boldsymbol{\beta}_j$, $\boldsymbol{\pi}_{-j} \leftarrow \boldsymbol{\pi} - \boldsymbol{\pi}_j$. Compute h_{-j}, ρ_{-j}, given $\boldsymbol{b}_j$ and $\boldsymbol{b}_j^2$ (precomputed). Do the local EP update, and compute $\boldsymbol{\pi}'$, $\boldsymbol{\beta}'$. Do damping. Write back $\boldsymbol{\pi}(V_j) \leftarrow \boldsymbol{\pi}_{-j} + \boldsymbol{\pi}'$, $\boldsymbol{\beta}(V_j) \leftarrow \boldsymbol{\beta}_{-j} + \boldsymbol{\beta}'$.

The local EP update is C code anyway. Maybe best to code a C routine which does sequential updates for a given list $[j_1, j_2, \dots]$. Here is a simple data structure. Keep EP parameters and rows of $\boldsymbol{B}$ in flat vectors: $[\boldsymbol{\pi}_j]$, $[\boldsymbol{\beta}_j]$, $[\boldsymbol{b}_j]$. Map j to V_j and start position in flat vectors. The C code could also optionally do selective damping in order to guarantee that $\pi_{-ki} \geq \varepsilon$ (see above).

5 Supported Potentials

In this section, we collect information about potentials which are (or will be) supported. At present, only univariate potentials $t(s)$, $s \in \mathbb{R}$, are supported. `EPPotentialFactory` collects registered potential classes in `C++`. Each class has a unique ID (nonnegative integer) and a unique name (string). The former are internal and maybe change. IDs and names are maintained by the factory class, a static method of which creates `EPScalarPotential` objects. Currently supported potentials are given in Table 1.

Name	Description
`Gaussian`	Gaussian (mean, variance)
`Laplace`	Laplace (mean, scale)
`Probit`	Probit, Gaussian c.d.f. (target, offset)
`Heaviside`	Heaviside step function (target, offset)
`Exponential`	Exponential distribution (scale)
`QuantRegress`	Quantile regression
`GaussMixture`	Gaussian mixture
`SpikeSlab`	Spike and slab

Table 1: Univariate potentials currently supported by the EP toolbox

Recall Section 3.2. The main service is

$$(\alpha, \nu, \log Z) \leftarrow \texttt{compMoments}(h_-, \rho_-, \eta = 1).$$

Here, $\eta \in (0,1]$ is a parameter for fractional EP (supported by some potentials only, $\eta = 1$ is the default supported by all). Here,

$$\hat{p}(s) = Z^{-1} t(s)^{\eta} N(s|h_-, \rho_-), \quad \mathrm{E}_{\hat{p}}[s] = h_- + \alpha\rho_-, \quad \mathrm{Var}_{\hat{p}}[s] = \rho_-(1 - \nu\rho_-).$$

In Section 6, we describe a generic implementation using numerical quadrature (adaptive or non-adaptive rules), which will allow for rapid prototyping with new potentials, and is also useful for debugging specific code. Moreover, for some important potentials (for example, Poisson likelihood with common rate functions, see Section 6.4), EP updates are not analytically tractable. If such quadrature implementations are used in the future, we will implement lookup table support (multilinear interpolation).

5.1 Design

EP (local) update services are provided by potential managers, which maintain a set of potential objects doing the work. For simplicity, both potential and manager objects are re-created on demand, so have to be light-weight. A potential object is configured by hyperparameters, and can optionally be annotated by an additional object (the "annotation") of arbitrary type. The hyperparameter vector consists of *construction parameters* (which have to form the prefix) and others, either of the two parts can be empty. `EPPotentialFactory` exports methods both for construction with full parameters and default construction. For the latter, only construction parameters (and annotation) have to be passed, the remaining parameters are set to valid default values. Most subclasses of `EPScalarPotential` neither have construction parameters nor an annotation, their objects can be default-constructed without parameters. An example for a construction parameter is the number L of mixture components in `EPPotGaussMixture`. This example shows that the number of hyperparameters need not be fixed for a class, in which case it has to be a function of construction parameters. In our implementation, construction parameters (if any) must come before normal parameters.

Annotation objects are used if parts of a potential object have to remain persistent, because they are expensive to re-create and/or because they are shared between different potential objects beyond block boundaries. For example, a potential object using numerical quadrature (see Section 6) has a `QuadratureService` object with working buffer, which is typicalled shared between several quadrature potential objects. Annotation objects have to be created independently and beforehand, they are made persistent by wrapping them in Python or Matlab objects. Formally, they are passed as additional construction parameter of type `void*`, which is `NULL` if the potential object is not annotated.

A design object has to export EP update services. For basic potentials (see below), updates are analytically tractable, and the user provides a complete implementation. For others, they are done by numerical quadrature and root finding code (see Section 6), and the user only provides simple code to evaluate $-\log t(s)$ and derivatives. In these cases, we use a `hasA` (instead of `isA`) relationship. The base class for potentials is `EPScalPotentialBase`, declaring get/set methods for parameters. Its child `EPScalarPotential` declares EP update services, and potentials with tractable updates are children of the latter. Another child of `EPScalPotentialBase` is `QuadraturePotential`, the base class for potentials supported by quadrature. The `hasA` bridge is given by `EPPotQuadrature`, a child of `EPScalarPotential` with a member of type `QuadraturePotential`. In this case, the user only has to implement the basic `QuadraturePotential` methods.

A potential manager (class `PotentialManager`) maintains a set of potentials. The subclass `ContainerPotManager` organizes them in a number of blocks. Within each block, the `EPScalarPotential` object is shared among all potentials. Each hyperparameter can either by shared (tied) or individual. Sharing of hyperparameters across blocks is not supported (except via shared annotations). Construction parameters and annotation (if any) are shared across the block (they are tied to the potential object).

5.2 Gaussian

$$t(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-s)^2}{2\sigma^2}}, \quad \sigma^2 > 0.$$

Parameters: y, $\sigma^2 > 0$. Fractional updates supported. Potential is log-concave.

$$\nu = \frac{1}{\rho_- + \sigma^2/\eta}, \quad \alpha = \nu(y - h_-), \quad \log Z = -\frac{1}{2}\left(\nu(y-h_-)^2 + \log(2\pi\eta/\nu)\right).$$

Note that EP in coupled mode does not run updates on Gaussian potentials, as this does not change anything.

5.3 Probit, Heaviside

$$t_{\text{probit}}(s) = \Phi(y(s+s_o)), \quad \Phi(x) = \int_{-\infty}^{x} N(t|0,1)\,dt, \quad y \in \{\pm 1\}$$
$$t_{\text{heavi}}(s) = \mathrm{I}_{\{y(s+s_o)\ge 0\}}$$

Parameters: $y \in \{\pm 1\}$, s_o. No fractional updates. Potential is log-concave.

Implementations are almost identical, `Probit` tolerates $\rho_- \approx 0$ better. Let $\rho_o = 1$ for `Probit`, $\rho_o = 0$ for `Heaviside`.

$$\log Z = \log \Phi(z), \quad z = \frac{yh_-}{\sqrt{\rho_- + \rho_o}}, \quad \alpha = \frac{yh(-z)}{\sqrt{\rho_- + \rho_o}}, \quad \nu = \alpha\left(\alpha + \frac{h_-}{\rho_- + \rho_o}\right),$$
$$h(-z) = \frac{N(z|0,1)}{\Phi(z)}$$

Here,

$$F(x) := \log(1 - \Phi(x)), \quad h(x) = \frac{N(x|0,1)}{1 - \Phi(x)}$$

are implemented in a robust manner (see Section 5.6). $h(x)$ is called Hazard function in statistics.

5.4 Laplace. Quantile Regression

The Laplace (or double exponential) potential is

$$t_{\text{lapl}}(s) = \frac{\tau}{2} e^{-\tau|y-s|}, \quad \tau > 0.$$

Parameters are y, $\tau > 0$. Fractional updates supported. Potential is log-concave.

This potential is a special case of the more general quantile regression potential [1]:

$$t(s) = \tilde{t}(\xi(y-s)), \quad \tilde{t}(r) = e^{-\kappa[r]_+ - (1-\kappa)[-r]_+}, \quad \xi > 0, \ \ \kappa \in (0,1),$$

where $[x]_+ := x\mathrm{I}_{\{x\ge 0\}}$. Parameters are y, $\xi > 0$, $\kappa \in (0,1)$. Here, the argument $r = \xi(y-s)$ is the scaled residual for a data case, y being the observed target. Namely,

$$t_{\mathrm{lapl}}(s)^\eta = C\tilde{t}(\xi(y-s)), \quad \kappa = \frac{1}{2}, \ \xi = 2\eta\tau, \ C = (\tau/2)^\eta.$$

Here the equations for the quantile regression potential $t(s)$:

$$\begin{aligned}
&h_r = \xi(y-h_-), \quad \rho_r = \xi^2\rho_-, \quad \log I_{0,1} = \frac{1}{2}\kappa(\kappa\rho_r - 2h_r) + F\left(\kappa\rho_r^{1/2} - h_r\rho_r^{-1/2}\right), \\
&\log I_{0,2} = \frac{1}{2}(1-\kappa)((1-\kappa)\rho_r + 2h_r) + F\left((1-\kappa)\rho_r^{1/2} + h_r\rho_r^{-1/2}\right), \\
&\log Z = \log I_0 = \log\left(e^{\log I_{0,1}} + e^{\log I_{0,2}}\right), \quad q = \sigma\left(\log I_{0,2} - \log I_{0,1}\right), \\
&\alpha = \xi(\kappa - q), \quad \nu = \xi^2\left(\rho_r^{-1/2}\exp\left(\log N(h_r\rho_r^{-1/2}) - \log I_0\right) - q(1-q)\right).
\end{aligned} \tag{2}$$

Here,

$$\log N(x) := -\frac{1}{2}\left(x^2 + \log(2\pi)\right), \quad F(x) := \log(1-\Phi(x)).$$

Derivation

The derivation for the Laplace potential is from [6], which is generalized and cleaned up here. First, $r = \xi(y-s)$. We derive the update for the r variable (eliminating y, ξ), then comment below on the update for s. We can write

$$\tilde{t}(r) = \mathrm{I}_{\{r\ge 0\}}e^{-\kappa r} + \mathrm{I}_{\{-r\ge 0\}}e^{-(1-\kappa)(-r)}.$$

When writing $\mathrm{E}[f(r)]$, we mean $r \sim N(h,\rho)$, derived from $q_-(s) = N(s|h_-,\rho_-)$. Below, we also write $\mathrm{E}[g(n)]$, where $n \sim N(0,1)$. First,

$$Z = I_0 = I_{0,1} + I_{0,2}, \quad I_{0,1} = \mathrm{E}\left[\mathrm{I}_{\{r\ge 0\}}e^{-\kappa r}\right], \quad I_{0,2} = \mathrm{E}\left[\mathrm{I}_{\{-r\ge 0\}}e^{-(1-\kappa)(-r)}\right].$$

Now, $e^{-\kappa r}N(r|h,\rho) = C_1 N(r|h_1,\rho)$ for some constant C_1 and mean h_1, but the same variance ρ. Some algebra gives

$$\begin{aligned}
&e^{-\kappa r}N(r|h,\rho) = C_1 N(r|h_1,\rho), \quad e^{-(1-\kappa)r}N(r|-h,\rho) = C_2 N(r|h_2,\rho), \\
&h_1 = h - \kappa\rho, \quad h_2 = -h - (1-\kappa)\rho, \quad C_k = \exp\left(\frac{1}{2\rho}(h_k^2 - h^2)\right), \\
&\log C_1 = \frac{1}{2}\kappa(\kappa\rho - 2h), \quad \log C_2 = \frac{1}{2}(1-\kappa)((1-\kappa)\rho + 2h).
\end{aligned}$$

Note that we use $-h$ in the "2" expressions, because the argument is $-r$ in $I_{0,2}$, and $-r \sim N(-h,\rho)$. In the sequel, we denote $\mathrm{E}_k[f(r)]$ where $r \sim N(h_k,\rho)$. Also, define

$$\gamma_k := -h_k\rho^{-1/2}, \quad \gamma_1 = \kappa\rho^{1/2} - h\rho^{-1/2}, \quad \gamma_2 = (1-\kappa)\rho^{1/2} + h\rho^{-1/2},$$

noting that $\mathrm{E}_k[\mathrm{I}_{\{r\geq 0\}}] = \mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}]$, where $n \sim N(0,1)$. With this preparation:

$$I_{0,k} = C_k\mathrm{E}_k[\mathrm{I}_{\{r\geq 0\}}] = C_k\mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}] = \exp\left(\log C_k + F(\gamma_k)\right), \quad F(x) := \log(1-\Phi(x)).$$

Here, $\Phi(x)$ is the cumulative distribution function of $N(0,1)$. It is very important to implement $F(x)$ properly, in order to avoid bad numerical trouble. What we compute is $\log I_{0,k}$, $k=1,2$, from which $\log I_0 = \log Z$ is computed by pulling out the maximum of the two.

Next,

$$I_1 = \mathrm{E}[r\tilde{t}(r)] = I_{1,1} - I_{1,2}, \quad I_{1,k} = C_k\mathrm{E}_k\left[\mathrm{I}_{\{r\geq 0\}}r\right]$$

and

$$I_{1,k} = C_k\mathrm{E}_k\left[\mathrm{I}_{\{n\geq\gamma_k\}}(h_k + \rho^{1/2}n)\right] = h_kI_{0,k} + \rho^{1/2}C_k\mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}n].$$

From [6]: $\mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}n] = N(\gamma_k)$, where we write $N(x)$ short for $N(x|0,1)$, the standard normal density. Also,

$$C_kN(\gamma_k) = N(\gamma), \quad \gamma := h\rho^{-1/2},$$

which does not depend on $k=1,2$. Then, $I_{1,k} = h_kI_{0,k} + \rho^{1/2}N(\gamma)$ and

$$\hat{h} = \frac{I_1}{I_0} = \frac{h_1I_{0,1} - h_2I_{0,2}}{I_0} = h + \rho\alpha,$$

where

$$\alpha = \frac{(1-\kappa)I_{0,2} - \kappa I_{0,1}}{I_0} = q - \kappa, \quad q := I_{0,2}/I_0 = \sigma\left(\log I_{0,2} - \log I_{0,1}\right).$$

Here, $\sigma(x) := 1/(1+e^{-x})$ is the logistic sigmoid function. Note that $q \in (0,1)$.

Next,

$$I_2 = \mathrm{E}[r^2\tilde{t}(r)] = I_{2,1} + I_{2,2}, \quad I_{2,k} = C_k\mathrm{E}_k\left[\mathrm{I}_{\{r\geq 0\}}r^2\right],$$

where

$$I_{2,k} = C_k\mathrm{E}_k\left[\mathrm{I}_{\{r\geq 0\}}r^2\right] = C_k\mathrm{E}\left[\mathrm{I}_{\{n\geq\gamma_k\}}(h_k+\rho^{1/2}n)^2\right] = h_k^2I_{0,k} + 2h_k\rho^{1/2}N(\gamma) + \rho C_k\mathrm{E}\left[\mathrm{I}_{\{n\geq\gamma_k\}}n^2\right].$$

From [6]: $\mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}n^2] = \mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}] + \gamma_kN(\gamma_k)$ and $C_k\mathrm{E}[\mathrm{I}_{\{n\geq\gamma_k\}}n^2] = I_{0,k} + \gamma_kN(\gamma)$, so that

$$I_{2,k} = (h_k^2 + \rho)I_{0,k} + h_k\rho^{1/2}N(\gamma),$$

where we used that $\rho\gamma_k = -h_k\rho^{1/2}$. Define

$$R := \frac{N(\gamma)}{I_0} = \exp\left(\log N(h\rho^{-1/2}) - \log I_0\right),$$

then

$$\frac{I_2}{I_0} = \rho + \frac{h_1^2I_{0,1} + h_2^2I_{0,2}}{I_0} + \rho^{1/2}(h_1+h_2)R.$$

Here, $h_1 + h_2 = -\rho$, and with some algebra:

$$h_1^2I_{0,1} + h_2^2I_{0,2} = h^2I_0 + 2h\rho\alpha I_0 + \rho^2\left(\kappa^2I_{0,1} + (1-\kappa)^2I_{0,2}\right).$$

Plugging this in, and using $\hat{h} - h = \rho\alpha$:

$$\frac{I_2}{I_0} = \rho + h^2 + 2h(\hat{h}-h) + \rho^2\eta - \rho^{3/2}R, \quad \eta := \kappa^2(1-q) + (1-\kappa)^2q.$$

Then,

$$\hat{\rho} = \frac{I_2}{I_0} - \hat{h}^2 = \rho - (\hat{h} - h)^2 + \rho^2 \eta - \rho^{3/2} R = \rho(1 - \nu\rho).$$

Using that $\hat{h} - h = \rho\alpha$, we have (some algebra):

$$\nu = \alpha^2 - \eta + \rho^{-1/2} R = \rho^{-1/2} R - q(1 - q).$$

As a sanity check, we compared these results for the Laplace case ($\kappa = 1/2$) to the expressions from [6]: they match.

Since $r = \xi(y - s)$ is a linear function, we can get the desired EP update w.r.t. s by (a) feeding in the cavity moments $h_r = \xi(y - h_-)$, $\rho_r = \xi^2 \rho_-$ (for h, ρ above), and (b) $\alpha = -\xi\alpha_r$, $\nu = \xi^2 \nu_r$. Plugging everything together, we obtain (2).

5.5 Mixture of Gaussian. Spike and Slab

The *mixture of Gaussians* potential is

$$t(s) = \sum_{l=1}^{L} p_l N(s|0, v_l).$$

The parameters are c_l, $l = 1, \dots, L - 1$ and v_l, $l = 1, \dots, L$, where

$$p_l = \frac{e^{c_l}}{\sum_k e^{c_k}}, \quad c_L = 0.$$

L is a construction parameter, so the complete hyperparameter vector is $[L, c_1, \dots, c_{L-1}, v_1, \dots, v_L]^T$. We assume that $v_l > 0$ for all l in the general derivation. However, everything goes through with $v_1 = 0$: this special case is treated below. The cavity distribution is $q_-(s) = N(s|\mu_-, \rho_-)$. We use its natural parameters $\pi_- = 1/\rho_-$, $\beta_- = \mu_-/\rho_-$ here. In fact, as noted at the end of this section, the local EP update can be done in certain cases if $\pi_- \le 0$, and in general it can be more robust to compute π_- directly instead of going through ρ_-. In order to support such a special treatment in the future, we assume that the input is (π_-, β_-). Also, we work with

$$\hat{p}(s) = \hat{Z}^{-1} t(s) e^{\kappa_-(s)}, \quad \kappa_-(s) := \beta_- s - \frac{1}{2}\pi_- s^2.$$

Our current implementation (which requires a proper cavity marginal) converts $(h_-, \rho_-) \to (\pi_-, \beta_-)$ at input, then

$$\log Z = \log \hat{Z} - \frac{1}{2}\left((h_-)^2/\rho_- + \log(2\pi\rho_-)\right)$$

at return.

Define

$$\rho_l = \frac{v_l}{1 + \pi_- v_l}, \quad \mu_l = \beta_- \rho_l.$$

Also,

$$Z_l = p_l \int e^{\kappa_-(s)} N(s|0, v_l)\, ds = p_l (\rho_l / v_l)^{1/2} e^{\frac{1}{2}\beta_-^2 \rho_l},$$

using that $\mu_l^2/\rho_l = \beta_-^2\rho_l$. Then, $\hat{Z} = \sum_l Z_l$. We comment on how to compute $\log\hat{Z}$ below. Define $r_l := Z_l/\hat{Z}$. Then,

$$\hat{p}(s) = \sum_l r_l N(s|\mu_l,\rho_l) = \mathrm{E}_r\left[N(s|\beta_-\rho_l,\rho_l)\right].$$

First,

$$\hat{\mu} = \beta_- A, \quad A := \mathrm{E}_r[\rho_l].$$

If $t = s - \hat{\mu}$, then

$$\hat{p}(t) = \mathrm{E}_Q\left[N(t|\beta_-(\rho_l - A),\rho_l)\right],$$

and

$$\hat{\rho} = \mathrm{E}_{\hat{p}}[t^2] = \mathrm{E}_r\left[\rho_l + \beta_-^2(\rho_l - A)^2\right].$$

Here is how to compute α, ν. Define

$$z_l := \frac{1}{1+\pi_- v_l} = 1 - \rho_l\pi_-, \quad \tilde{A} := \mathrm{E}_r[z_l].$$

Note that $A/\rho_- = 1 - \tilde{A}$. Then,

$$\hat{\mu} = \mu_- A/\rho_- = \mu_-(1-\tilde{A}) \quad \Rightarrow \quad \alpha = -\beta_-\tilde{A}.$$

Also, $(\rho_l - A)^2 = \rho_-^2(z_l - \tilde{A})^2$, so that

$$\hat{\rho} = \rho_-(1-\tilde{A}) + \beta_-^2\rho_-^2\mathrm{Var}_r[z_l] \quad \Rightarrow \quad \nu = \tilde{A}\pi_- - \beta_-^2\mathrm{Var}_r[z_l] = \tilde{A}\pi_- - \beta_-^2\mathrm{E}_r[z_l^2] + \alpha^2.$$

In practice, we use `logsumexp` accumulation for $\log\hat{Z}$ and $\log\mathrm{E}_r[z_l^k]$, $k = 1,2$, the latter are based on values $\log Z_l + k\log z_l$. In a first loop, we compute the $\log Z_l$ and determine the maxima for the accumulators. In the second loop, we do the `logsumexp` accumulations, subtracting $\log\hat{Z}$ at the end.

Finally, a MoG potentials with component means different from 0 can be implemented similarly, but is not part of the toolbox at present.

A fully general MoG potential has hyperparameters for component variances and means, apart from $[c_l]$. Given that approximate integration over variance hyperparameters in Gaussian potentials is implemented (Section 7.1), this could easily be adapted to integrating out mean and variance hyperparameters of the MoG potential.

Spike and Slab

The *spike and slab* potential (in its simplest form) is a special case of the MoG potential, where $L = 2$ and $v_1 = 0$:

$$t(s) = (1-p)\delta_0(s) + pN(s|0,v), \quad c = \log\frac{p}{1-p}, \ v > 0.$$

Its parameter vector is $[c,v]^T$. Mapping to the MoG case, we have that $z_1 = 1$, $p_1 = \sigma(-c)$, $p_2 = \sigma(c)$, where $\sigma(x) = 1/(1+e^{-x})$. Also, $A = r_2\rho_2$, $\tilde{A} = 1 + r_2(z_2 - 1)$, and

$$\begin{aligned}\mathrm{Var}_r[z_l] &= 1 + r_2(z_2^2 - 1) - (1 + r_2(z_2 - 1))^2 = r_2\left(z_2^2 - 1 - 2(z_2 - 1) - r_2(z_2 - 1)^2\right) \\ &= r_2(1-r_2)(z_2 - 1)^2.\end{aligned}$$

Again, the technique developed in Section 7.1 could be used to integrate out the slab variance hyperparameter v.

Avoiding the Computation of Cavity Marginals

For difficult potentials like MoG and spike&slab, it can be a good idea to avoid computing cavity marginals explicitly, or at least to shoot for their natural parameters π_-, β_- directly. This is easy to do in the coupled modes. If the current marginal is $q(s) = N(s|\mu, \rho)$ and the EP parameters are π, β, then the natural parameters of $q_-(s)$ are given by

$$\pi_- = 1/\rho - \pi, \quad \beta_- = \mu/\rho - \beta.$$

The cavity marginal is a proper Gaussian iff $\pi_- > 0$, and we may even restrict $\pi_- \geq \varepsilon > 0$. On the other hand, apart from the computation of $\log Z$, the local EP update for a MoG potential goes through if $\pi_- + 1/(\max_l v_l) > 0$, so π_- can even be negative.

In the factorized mode, computing the well-defined cavity marginal $q_-(s)$ seems essential. We can improve numerical stability by computing the natural parameters π_-, β_- directly. Recall that

$$h_{-j} = \sum_i b_{ji}\beta_{-ji}/\pi_{-ji}, \quad \rho_{-j} = \sum_i b_{ji}^2/\pi_{-ji}.$$

We require that $\phi_{-ji} > 0$, but it may still be very small. Let $\eta := \min_i \pi_{-ji}/b_{ji}^2$. Then,

$$\pi_{-j} = \frac{\eta}{\sum_i (b_{ji}^2/\pi_{-ji})\eta}, \quad \beta_{-j} = \frac{\sum_i (b_{ji}\beta_{-ji}/\pi_{-ji})\eta}{\sum_i (b_{ji}^2/\pi_{-ji})\eta}.$$

Here, we accumulate η and the denominator together in the usual way, avoiding ratios of large numbers.

5.6 Numerical Routines

In this section, we collect comments on numerical routines required for EP update code. At least the most important ones should be implemented or directly taken from `netlib`, so to avoid unnecessary dependencies.

Error Function

Write $N(x)$ for $N(x|0,1)$, the density of the standard normal distribution, and $\Phi(x)$ for its cumulative distribution function: $\Phi(x) = \int_{-\infty}^{x} N(t)\,dt$, $N(x) = d\Phi(x)/dx$. Define

$$F(x) = \log(1 - \Phi(x)), \quad h(x) = \frac{N(x)}{1 - \Phi(x)}.$$

These are required for a number of potentials. The standard implementation of $1 - \Phi(x)$ in `Matlab` and elsewhere, related to `erfc`, is due to [2]. The complementary error function erfc is related to $\Phi(x)$ as:

$$1 - \Phi(x) = \frac{1}{2}\mathrm{erfc}(x/\sqrt{2}).$$

Assume for now that $x \geq 0$. Cody's implementation (described in [2], Fortran code available at `www.netlib.org/specfun/erf`) splits the argument range $[0, \infty)$ into three intervals, say 1, 2, 3. Part 1: $\tilde{x} = x/\sqrt{2} \geq 4$. Then:

$$\mathrm{erfc}(\tilde{x}) \approx \frac{e^{-\tilde{x}^2}}{\tilde{x}} \left(\pi^{-1/2} + \tilde{x}^{-2} R_1(\tilde{x}^{-2})\right),$$

where $R_1(y)$ is a rational function of polynomials of degree five. Some algebra gives

$$1 - \Phi(x) \approx \frac{N(x)}{x} Q_1(x), \quad Q_1(x) := 1 + \sqrt{\pi} y R_1(y), \quad y = \frac{2}{x^2}.$$

Note that $Q(x) > 1$. Then:

$$F(x) \approx -\frac{1}{2}\left(x^2 + \log(2\pi)\right) + \log(Q_1(x)/x), \quad h(x) \approx \frac{x}{Q_1(x)}.$$

Part 2: $\tilde{x} = x/\sqrt{2} \in [0.46875, 4)$. Then:

$$\operatorname{erfc}(\tilde{x}) \approx e^{-\tilde{x}^2} R_2(\tilde{x}),$$

where $R_2(\tilde{x})$ is a rational function. Then,

$$1 - \Phi(x) \approx \frac{N(x)}{x} Q_2(x), \quad Q_2(x) := \sqrt{\pi}\tilde{x} R_2(\tilde{x}), \quad \tilde{x} = \frac{x}{\sqrt{2}}.$$

$F(x)$ and $h(x)$ are defined as in part 1. Part 3: $\tilde{x} = x/\sqrt{2} \in [0, 0.46875)$. Then:

$$\operatorname{erfc}(\tilde{x}) \approx 1 - \tilde{x} R_3(\tilde{x}^2),$$

where $R_3(\tilde{x})$ is a rational function. Then,

$$1 - \Phi(x) \approx \frac{1}{2}\left(1 - \tilde{x} R_3(\tilde{x}^2)\right), \quad \tilde{x} = x/\sqrt{2}.$$

And:

$$F(x) = \log\left(1 - \tilde{x} R_3(\tilde{x}^2)\right) - \log 2, \quad h(x) = \frac{N(\tilde{x})}{\frac{1}{2}(1 - \tilde{x} R_3(\tilde{x}^2))}, \quad \tilde{x} = x/\sqrt{2}.$$

For $x < 0$, the computation of $F(x)$, $h(x)$ is uncritical. We have that $1 - \Phi(x) = 1 - (1 - \Phi(-x))$, so the part is determined from $-x > 0$. For part 1 and 2, we compute $F(x)$, $h(x)$ directly from the $1 - \Phi(-x)$ approximation. For part 3, note that $1 - \Phi(x) \approx \frac{1}{2}(1 - \tilde{x} R_3(\tilde{x}^2))$ is still valid, so the code remains the same.

6 Generic EP Updates: Adaptive Quadrature

In this section, we detail a framework for generic local EP updates (see Section 5) by way of numerical quadrature. While quadrature may be required in other contexts as well (see Section 7.1), we restrict ourselves to the task of approximating $\log Z$, α, ν, given the cavity marginal $q_-(s) = N(s|h_-, \rho_-)$. The role for this framework lies in rapid prototyping: if a potential is really used, a specific implementation typically is more accurate and more efficient, but can take some effort to develop. The generic framework is not optimized for efficiency. We use foreign adaptive quadrature code, which does not allow for reuse of function evaluations. Nonsmooth integrands are split into parts. Typically, the mode of the integrand needs to be found in order to improve the accuracy. Given that the generic framework is used in time-critical applications, we plan to develop a fast lookup table interpolation solution in order to save on quadrature calls.

In the sequel, we assume that $t(s)$ is defined on all of $\mathbb{R}$, but the domain of integration may be restricted:

$$Z = \int_a^b t(s) q_-(s)\, ds, \quad -\infty \le a < b \le \infty.$$

Below, we write $g(s) = t(s)q_-(s) = e^{-h(s)}$. At least for now, we assume that $g(s)$ is bounded on $[a, b]$. Apart from $\log Z$, we target

$$I_k = \mathrm{E}_{\hat{p}}\left[s^k\right] = Z^{-1} \int_a^b s^k t(s) q_-(s)\, ds, \quad k = 1, 2.$$

There are (at least) two major issues we have to deal with before calling adaptive quadrature code (as implemented in `quadpack`). First, $t(s)$ may not be smooth in $[a, b]$. Our framework allows to partition $[a, b]$ into a small number of adjacent intervals, such that $t(s)$ is smooth (at least twice continuously differentiable) on either. Second, if $t(s)$ and $q_-(s)$ have little overlap in $[a, b]$, Z can become very small, and any sensible quadrature routine might underflow. This issue could probably be dealt with by changing the quadrature code, so to work in the log domain and to make sure of `logsumexp`, but this is not an option. We support a heuristic which should work for many potentials, but not in general [4]. Essentially, $g(s)$ is fitted by a Gaussian using the Laplace approximation, and the integration variable is transformed to make this a standard normal. A drawback of this procedure is that $g(s)$ should be unimodal (even better: log-concave), and its mode has to be found. Importantly, our framework is modular and can be configured by the user. The integration domain can be partitioned. The computations on each subinterval make use of the same code. We provide a 1D Newton solver for mode finding, but allow the user to provide special solutions, for example if the mode can be determined analytically. Finally, different quadrature routines can be called on the transformed problems. We support expensive adaptive quadrature, but also allow for cheaper Gauss-Hermite quadrature.

6.1 Support for Piecewise-smooth Integrands

Suppose that $[a, b]$ is partitioned into adjacent intervals $A_1, \dots, A_J$. If $J = 1$, then $A_1 = [a, b]$, and nothing has to be done. Suppose that $J > 1$, and define

$$Z^{(j)} = \int_{A_j} t(s) q_-(s)\, ds, \quad I_k^{(j)} = (Z^{(j)})^{-1} \int_{A_j} s^k t(s) q_-(s)\, ds.$$

Then,

$$\log Z = \log \sum_j e^{\log Z^{(j)}}, \quad I_k = \sum_j r_j I_k^{(j)}, \quad r_j = \frac{Z^{(j)}}{Z} = e^{\log Z^{(j)} - \log Z}.$$

The different $(\log Z^{(j)}, I_1^{(j)}, I_2^{(j)})$ can be computed independently of each other, using the same code. In the sequel, we may assume that $t(s)$ is smooth on $[a, b]$.

Note that in our first implementation, partitioning is not fully supported. The user has to provide waypoints in (a, b) where $\log t(s)$ is not twice continuously differentiable. This information is used in the automatic variable transformation, and it is passed to the quadrature code. At present, we use expensive `quadpack` routines which can deal with non-smooth integrands automatically. If $[a, b]$ is a finite interval, we can speed things up by passing the waypoints to the routine. However, if $a = -\infty$ or $b = \infty$, this information cannot be passed (for whatever reason).

In the future, if quadrature is used in time-critical situations, we plan to implement explicit partitioning along fast non-adaptive rules on each interval.

6.2 Variable Transformation by Laplace Approximation

Recall that $g(s) = t(s)q_-(s) = e^{-h(s)}$. The heuristic we use in order to linearly transform the integration variable has been proposed in the context of Gauss-Hermite quadrature, to find an appropriate Gaussian weighting function [4]. The idea is to fit $h(s)$ to second order at its minimum point (Taylor approximation), which amounts to a Laplace approximation of the integral. In order for this to work, $g(s)$ should be unimodal. It works best if $g(s)$ is log-concave (i.e., $h(s)$ is convex), in which case its mode can be found by convex minimization. Let

$$s_* = \operatorname*{argmin}_s h(s), \quad \sigma^2 = 1/h''(s_*).$$

Note that s_* is the argmin for the *proximal mapping* of $-\rho_- \log t(s)$. We substitute $s = s_* + \sigma x$, so that

$$\log Z = -h(s_*) + \log\sigma - \frac{1}{2}\log(2\pi\rho_-) + \log\int_{\tilde{a}}^{\tilde{b}} \exp\left(-h(s_* + \sigma x) + h(s_*)\right)\,dx,$$
$$\tilde{a} = (a - s_*)/\sigma,\ \tilde{b} = (b - s_*)/\sigma.$$

More general, the transformation is well-defined if s_* is a local minimum point of $h(s)$ and $h''(s_*) > 0$. We call a quadrature routine for

$$\tilde{Z} = \int_{\tilde{a}}^{\tilde{b}} \tilde{g}(x)\,dx, \quad \tilde{g}(x) := e^{-h(s_*+\sigma u)+h(s_*)},$$

from which $\log Z$ is obtained. Further quadrature calls give

$$\tilde{I}_k = \tilde{Z}^{-1}\int_{\tilde{a}}^{\tilde{b}} x^k \tilde{g}(x)\,dx,$$

from which α, ν can be determined:

$$\alpha = \frac{\sigma\tilde{I}_1 + s_* - h_-}{\rho_-}, \quad \nu = \frac{1 - \sigma^2(\tilde{I}_2 - \tilde{I}_1^2)/\rho_-}{\rho_-}.$$

In the most generic case, the user supplies code for $s \mapsto (l(s), l'(s), l''(s))$, where $l(s) = -\log t(s)$. s_* is found by 1D Newton minimization. The user can override this part by code for finding s_* directly. It makes sense to store the current s_* for each potential and use it as a starting value in the next round.

Note that the variable transform does not depend on the interval $[a, b]$. In particular, s_* may well lie outside of this interval. If several intervals A_j are used, s_* and σ^2 have to be determined only once. If s_* falls onto a critical point (endpoint of a A_j), our implementation uses $\sigma^2 = \rho_-$ instead of evaluating $l''(s_*)$. At present, our generic implementation uses a 1D Newton solver in order to find s_*, which requires that $h(s)$ is twice differentiable everywhere (apart from being unimodal). Solvers which can cope with critical points, may be added in the future.

If some expensive adaptive quadrature code is used, running the Laplace transformation up front may be overkill. We could evaluate $h(s)$ at a few chosen points, including h_-, and pick the minimizer for s_*, then either rescale by $\sigma^2 = \rho_-$ or not at all. However, the Laplace transformation can be combined with a cheap non-adaptive rule, such as Gauss-Legendre. We would write

$$Z = e^{-h(s_*)}(\sigma^2/\rho_-)^{1/2} \int_{\tilde{a}}^{\tilde{b}} e^{h(s_*)+\frac{1}{2}x^2-h(s_*+\sigma x)} N(x|0,1)\, dx,$$

where $N(x|0,1)$ is the weighting function. Our implementation allows for different quadrature code to be combined with transformations.

6.3 Design

Recall Section 5.1. The user has to implement the `QuadraturePotential` interface (child of `EPScalPotentialBase`) for any quadrature implementation. Children of `QuadraturePotential` and `EPScalarPotential` do separate things. The former implement low-level methods to drive quadrature code (evaluation of $-\log t(s)$, etc.), the latter implement high-level EP update services. The base class for the latter is `EPPotQuadrature`, child of `EPScalarPotential` with a `QuadraturePotential` member.

Children of `EPPotQuadrature` make use of quadrature services, provided by implementations of `QuadratureService`. At present, this interface is very simple, but it will be specialized to support weighting functions on demand. The `QuadratureService` object is an annotation (see Section 5.1): it maintains working memory and can be shared among different potential objects.

`EPPotQuadLaplaceApprox` (child of `EPPotQuadrature`) implements the Laplace variable transformation from Section 6.2. It maintains a `QuadPotProximal` object, which extends `QuadraturePotential` by a proximal map method. In smooth, log-concave cases, the subclass `QuadPotProximalNewton` can be used, a 1D Newton implementation of the proximal map just based on `QuadraturePotential` methods. Examples for implementations are `EPPotPoissonExpRate` and `EPPotDebugQuadLaplace` (implement proximal map directly) and `EPPotPoissonLogisticRate` (derived from `QuadPotProximalNewton`).

The `hasA` relationship in `EPPotQuadrature` makes it easy for a user to extend the hierarchy. For rapid prototyping with a smooth (and ideally log-concave) potential, it suffices to implement `QuadraturePotential` and run experiments with (expensive) adaptive quadrature code. Specific code for the proximal map can be added by deriving from `QuadPotProximal`. Finally, EP update services can be specialized themselves, and the implementation can be moved out of the quadrature framework.

6.4 Example: Poisson Potential

The *Poisson* potential is

$$t(s) = \frac{1}{y!}\lambda(s)^y e^{-\lambda(s)}, \quad y \in \mathbb{N},\ \lambda(s) > 0.$$

Commonly used rate functions are (A) $\lambda(s) = e^s$ and (B) $\lambda(s) = \log(1 + e^s)$. The Poisson potential $t(s)$ is log-concave for both choices. While (A) is simpler to handle, (B) has the advantage of $\lambda(s)$ only growing linearly as $s \to \infty$.

In either case, to our knowledge, EP updates are not analytically tractable for any $y \in \mathbb{N}$. Note that (B) with $y = 0$ corresponds to $t(s) = (1 + e^s)^{-1} = \sigma(-s)$, the logistic potential for binary classification. We have:

$$h(s) = -y \log \lambda(s) + \lambda(s) + \frac{1}{2\rho_-}(s - h_-)^2.$$

For case (A):

$$h'(s) = -y + e^s + (s - h_-)/\rho_-.$$

If $\tilde{s} = s + \log \rho_-$, then

$$h'(s) = 0 \quad \Leftrightarrow \quad e^{\tilde{s}} + \tilde{s} = h_- + y\rho_- + \log \rho_-.$$

This can be solved very easily by 1D Newton, without the need for any safety measures. Note that all parameters are combined in the right hand side.

For case (B), note that $\lambda'(s) = \sigma(s) = (1 + e^{-s})^{-1}$, $\sigma'(s) = \sigma(s)(1 - \sigma(s))$. Then,

$$h'(s) = \left(1 - \frac{y}{\lambda(s)}\right)\sigma(s) + (s - h_-)/\rho_-$$

and

$$h''(s) = \rho_-^{-1} + \sigma(1 - \sigma) + y(\sigma/\lambda)\left(\sigma - 1 + \sigma/\lambda\right).$$

Here, σ/λ is strictly decreasing, $\sigma/\lambda \to 1$ rapidly as $s \to -\infty$ and $\sigma/\lambda \approx s^{-1}$ as $s \to \infty$. We have that $h''(s) > \rho_-^{-1}$, so 1D Newton minimization should be safe. In this case, our generic Newton code should be used.

Initial Brackets

Our 1D Newton solver requires an initial bracket $[L, R]$ which is guaranteed to contain a root. In case (A), the function is $f(s) = e^s + s - a$, where we write s for $\tilde{s}$. If $a \le 1$, then $L = a - e^a$, $R = a$ is a bracket of width $e^a \le e$. If $a > 1$, $R = \log a$ gives $f(R) = \log a > 0$. The ansatz $L = (1 - e)\log a$, $e \in (0, 1)$, gives

$$f(L) = a\left(a^{-e} - 1\right) + (1 - e)\log a < a\left(a^{-e} - 1\right) + \log a.$$

Setting the r.h.s. equal to zero gives

$$e = \frac{-\log(1 - (\log a)/a)}{\log a}, \quad f(L) = -e \log a < 0.$$

Here, $e \approx 1/a$ as a gets large, so the bracket width is $\approx (\log a)/a$.

For case (B), the function is $f(s) = \rho_- h'(s)$. We have

$$f(s) = \rho_-\left(\sigma(s) - y\kappa(s)\right) + s - h_-, \quad \kappa(s) = \sigma(s)/\lambda(s).$$

Since $\kappa(s) > 0$, $\sigma(s) < 1$:

$$f(s) < \rho_- + s - h_- \quad \Rightarrow \quad f(L) < 0,\ L = h_- - \rho_-.$$

Next, $\kappa(s) < 1/s$ for $s > 0$. Assume for now that $y \neq 0$. For $s > a \geq 0$:

$$f(s) > \rho_- \left(\sigma(a) - y/s\right) + s - h_-.$$

Setting the r.h.s. to zero leads to the quadratic $s^2 - (h_- - \sigma(a)\rho_-)s - \rho_- y$, whose positive solution is

$$s_*(a) = \frac{1}{2}\left(h_- - \sigma(a)\rho_- + \sqrt{(h_- - \sigma(a)\rho_-)^2 + 4\rho_- y}\right).$$

If $y = 0$, then

$$f(s) > \sigma(a)\rho_- + s - h_- \quad \Rightarrow \quad s_*(a) = h_- - \sigma(a)\rho_-.$$

Here, we choose a as large as possible so that $s_*(a) > a$. We can always choose $a = 0$, for which $\sigma(a) = 1/2$. The right bracket end is $R = s_*(a)$. This bracket is small if $h_- - \rho_- > 0$, but can be overly large for negative h_- (at least if $y \neq 0$). On the other hand, R does not matter much in our algorithm.

6.5 Example: Negative Binomial

The negative binomial distribution over $y \in \mathbb{N}$ is an alternative to the Poisson distribution for "overdispersed" data. Note that $\mathrm{E}[y|s] = \mathrm{Var}[y|s] = \lambda(s)$ for the Poisson, while the variance for the negative binomial can be larger and is controlled by a dispersion parameter:

$$t(s) = \frac{\Gamma(r+y)}{\Gamma(y+1)\Gamma(r)}\left(\frac{r}{r+\lambda(s)}\right)^r \left(\frac{\lambda(s)}{r+\lambda(s)}\right)^y, \quad y \in \mathbb{N},\ \lambda(s) > 0,\ r > 0.$$

Here, $t(s) = P(y|\lambda(s), r)$. $r > 0$ is a dispersion parameter. Note that $\mathrm{E}[y|s] = \lambda(s)$ as for the Poisson, but $\mathrm{Var}[y|s] = \lambda(s) + \lambda(s)^2/r$, which grows with shrinking r. The density can also be parameterized in terms of $p(s) = \lambda(s)/(r + \lambda(s))$. In that case, if $r \in \mathbb{N} \setminus \{0\}$, y is distributed as the number of successes in a sequence of i.i.d. coin flips with success probability $p(s)$, stopping with the r-th failure.

As a function of λ,

$$-\log t(\lambda) = -y \log \lambda + (r+y)\log(r+\lambda)$$

is neither convex nor concave. It is convex on the left and concave on the right. If used with an exponential rate function $\lambda(s) = e^s$,

$$l(s) = -\log t(s) \doteq -ys + (y+r)\log\left(r + e^s\right), \quad l'(s) = -y + (y+r)\sigma(s - \log r).$$

$l'(s)$ is increasing, so $l(s)$ is convex and $t(s)$ is log-concave. Plugging this into our quadrature code, the Laplace transformation is based on

$$f(s) = \rho_-\left((y+r)\sigma(s - \log r) - y\right) + s - h_-.$$

Using that $\sigma(\cdot) \in (0,1)$, we obtain the initial bracket $L = h_- - \rho_- r$, $R = h_- + \rho_- y$.

If we use the logistic rate function $\lambda(s) = \log(1 + e^s)$, then $l(s)$ is not convex in general. In fact, $l(s)$ is convex on $(-\infty, \tilde{s}]$ and concave on $[\tilde{s}, \infty)$. On the other hand, the negative curvature of $l(s)$ is easily bounded below, so that $f(s)$ may still be convex. Even if $f(s)$ is not convex, its minimization should still be doable by way of a case distinction. Details to be worked out (only if this potential is really required).

6.6 Variable Transformation by Current Marginal

Another variable transform has been proposed in [8]. In the coupled modes, we have that $q_{-}(s) \propto q(s)e^{-\kappa(s)}$, where $\kappa(s) := \beta s - \frac{1}{2}\pi s^2$. In the factorized mode, let $q(s)$ be the Gaussian marginal obtained in the last recent EP update for $t(s)$, and define $e^{\kappa(s)} \propto q(s)/q_{-}(s)$. During later stages of EP, we can assume that $q(s)$ is a good Gaussian approximation to the *new* $\hat{p}(s) = Z^{-1}t(s)q_{-}(s)$. We can write

$$\hat{p}(s) = \hat{Z}^{-1}t(s)e^{-\kappa(s)}q(s).$$

If $q(s) = N(\mu, \sigma^2)$, we substitute $s = \mu + \sigma u$. The rest is the same as in Section 6.2. The advantage of this alternative is that no numerical minimization is required. On the other hand, as noted in [8], the approximation can be very poor if $q(s)$ is far from $\hat{p}(s)$, and this is bound to happen in early iterations. A generic implementation should not rely on this transformation alone.

One interesting use may be to use the Laplace transformation, but to initialize the 1D minimization algorithm at the mean of $q(s)$. At least in the coupled modes, this could save some memory.

7 Possible Extensions

In this section, we collect ideas and details for possible extensions.

7.1 Inference over Precision Hyperparameters

A first step towards supporting inference in hierarchical models is to allow for hyperparameters to be integrated out approximately. Short of an accurate solution for multi-dimensional quadrature, this has to be done on a case-by-case basis. Here, we discuss a case which occurs frequently in practice, and which incorporates some of the difficulties. Consider a Gaussian potential $t_j(s_j) = N(s_j|\mu, \tau^{-1})$, the hyperparameters being μ, τ (mean and precision). Typically, hyperparameters are shared by many potentials. The mean μ can already be treated in the framework above, by reparameterizing $s_j \leftarrow s_j - \mu$, so we can consider $t_j(s_j) = N(s_j|0, \tau^{-1})$.

We will add τ to the random variables of the model. The canonical distribution for inverse variances is the *Gamma* family:

$$\mathcal{G}(\tau|a, c) = \frac{c^a}{\Gamma(a)}\tau^{a-1}e^{-c\tau}, \quad a, c > 0.$$

Also,

$$\mathcal{G}^U(\tau|a, c) = \tau^a e^{-c\tau}$$

parameterizes ratios of Gamma densities. The backbone distribution will be

$$q(\boldsymbol{x}, \tau) = q(\boldsymbol{x})q(\tau),$$

where $q(\boldsymbol{x})$ is Gaussian as above, and $q(\tau)$ is Gamma. We restrict ourselves to the following simple setup. For an index subset G, we have that $t_j(s_j, \tau) = N(s_j|0, \tau^{-1})$ for all $j \in G$, while $t_j(s_j)$, $j \notin G$, are standard univariate potentials. More general, we can allow for a number of different variance hyperparameters, at no additional complexity. In this case, the backbone distribution must be factorized w.r.t. the hyperparameters, no matter what mode is used for $\boldsymbol{x}$.

One remark up front. EP updates with $N(s_j|0,\tau^{-1})$ could be done analytically if we used the Normal-Gamma family for the backbone distribution:

$$q(\boldsymbol{x},\tau) = \prod_i q(x_i|\tau)q(\tau),$$

where the $q(x_i|\tau)$ are conditional Gaussian. However, this leads to very complex updates for univariate potentials $j \notin G$, where EP updates have to be run inside quadrature code, and there is little hope for efficient computation. In contrast, the proposal here is *modular*: quadrature is only required for potentials $j \in G$. Since these are Gaussian, the quadrature problems are of the same form, and we can implement lookup tables for fast computation. Moreover, the rest of the code remains the same, so the extension can be implemented as add-on.

It is clear that both in the coupled and the factorized modes, we only have to deal with the local EP update on a potential $j \in G$, computing the marginal moments of

$$\hat{p}_j(s_j,\tau) \propto N(s_j|0,\tau^{-1})q_{-j}(s_j)q_{-j}(\tau).$$

In the factorized mode, the term approximations (messages) are

$$m_{ji}(x_i) = N^U(x_i|\beta_{ji},\pi_{ji}), \quad m_j(\tau) = \mathcal{G}^U(\tau|a_j,c_j),$$

while

$$q(x_i) \propto N^U(x_i|\beta_i,\pi_i), \quad q(\tau) = \mathcal{G}(\tau|a,c).$$

As above, we have

$$\beta_{-ji} = \beta_i - \beta_{ji},\ \pi_{-ji} = \pi_i - \pi_{ji},\ a_{-j} = a - a_j,\ c_{-j} = c - c_j,$$

and $q_{-j}(s_j) = N(h_{-j},\rho_{-j})$ is computed in the same way.

Local EP Update

Fix $j \in G$ and drop it from notation. We have $q_-(s) = N(h,\rho)$, $q_-(\tau) = \mathcal{G}(a,c)$, $t(s) = N(s|0,\tau^{-1})$. Note that we write h instead of h_-, etc., for simplicity. Then,

$$\hat{p}(s,\tau) \propto N(s|0,\tau^{-1})N(s|h,\rho)\mathcal{G}(\tau|a,c).$$

There are two options. Either we integrate analytically over s, then do numerical quadrature over τ, or the other way around. We choose the first option: the second results in an integral over Student t times Gaussian, which sounds painful. For the first, the integral over τ seems nicely behaved. First, we need $\hat{h} = \mathrm{E}[s]$, $\hat{\rho} = \mathrm{Var}[s]$. By a standard derivation:

$$N(s|0,\tau^{-1})N(s|h,\rho) \propto N(s|\kappa h,\kappa\rho)(\kappa\tau)^{\frac{1}{2}}e^{\frac{1}{2}\kappa\xi}, \quad \kappa = \frac{\rho\tau}{1+\rho\tau}, \quad \xi = \frac{h^2}{\rho}.$$

The desired moments are

$$\hat{h} = \mathrm{E}[\kappa]h, \quad \hat{\rho} = \mathrm{E}[\kappa]\rho + \mathrm{Var}[\kappa]h^2.$$

Note that $\kappa \in (0,1)$, so $|\hat{h}| < |h|$. Define

$$f_l(\kappa,\xi) = \kappa^{l+\frac{1}{2}} e^{-\frac{1}{2}(1-\kappa)\xi}.$$

Absorbing the $\tau^{\frac{1}{2}}$ term into the Gamma density, and dividing by the constant $e^{\frac{1}{2}\xi}$, we have

$$\hat{p}(\tau) \propto f_0(\kappa,\xi)\mathcal{G}(\tau|a+1/2,c).$$

Moments depend on four parameters. By reparameterizing $v = \rho\tau$:

$$\hat{p}(v) = Z^{-1} f_0(\kappa,\xi)\mathcal{G}(v|a+1/2,c/\rho), \quad \kappa = \frac{v}{1+v}.$$

Then

$$Z = \mathrm{E}_0[f_0(\kappa,\xi)], \quad \mathrm{E}[\kappa] = Z^{-1}\mathrm{E}_0[f_1(\kappa,\xi)], \quad \mathrm{E}[\kappa^2] = Z^{-1}\mathrm{E}_0[f_2(\kappa,\xi)],$$

where $\mathrm{E}_0[\cdot]$ is over $\mathcal{G}(v|a+1/2,c/\rho)$. The results depend on three parameters: a, c/ρ, $\xi = h^2/\rho$. It is unclear whether this can be reduced to two parameters. For not too large ξ, the integrand is well-behaved, since $\kappa \in (0,1)$.

Next, $q'(\tau)$ can be fit by first and second moments as well. Namely,

$$\mathrm{E}[\tau^l] = \rho^{-l} Z^{-1}\mathrm{E}_0\left[v^l f_0(\kappa,\xi)\right].$$

Absorbing v^l into the Gamma density:

$$\mathrm{E}[\tau^l] = \left(\prod\nolimits_{j=0}^{l-1} \frac{a+j+1/2}{c}\right) Z^{-1}\mathrm{E}_l[f_0(\kappa,\xi)],$$

where $\mathrm{E}_l[\cdot]$ is over $\mathcal{G}(v|a+l+1/2,c/\rho)$. The code (or lookup table) for computing Z can be used for all these moments. Now,

$$\frac{a}{c} = x_1 := \mathrm{E}[\tau], \quad \frac{a+1}{c} = x_2 := \frac{\mathrm{E}[\tau^2]}{\mathrm{E}[\tau]}.$$

Note that Z^{-1} drops out in the ratio for x_2, it then has the same form as $x_1 = \mathrm{E}[\tau]$ (ratio of Gamma expectations over f_0). And

$$\hat{c} = \frac{1}{x_2 - x_1}, \quad \hat{a} = \frac{x_1}{x_2 - x_1}.$$

It is important to make sure that the quadrature results are $O(1)$, neither too small nor too large. Recall that $Z = \mathrm{E}_0[f_0(\kappa,\xi)]$. By dividing through $e^{\frac{1}{2}\xi}$ above, we defined things so that $f_0(\kappa,\xi) \in (0,1)$. For moderate values of ξ, this integrand is very well-behaved, and $Z = O(1)$. If large values of ξ occur in practice, it will be important to analyze under which conditions Z becomes very small. Note that κ is strictly increasing with v, with $\kappa = 1/2$ for $v = 1$ (which means $\tau = \rho^{-1}$). Therefore, f_0 is smaller than $e^{-\xi/4}$ for $v < 1$, and small values of Z can arise if the Gamma distribution for v has much of its mass there. In fact, instead of dividing by $e^{\frac{1}{2}\xi}$ above, it may be better to divide by $e^{\frac{1}{2}r\xi}$, where r is optimized depending on the Gamma distribution.

A more principled approach would be to adapt the Laplace approximation from Section 6.2. Here, we need a local fit by a Gamma distribution. Note that $f_l(\kappa, \xi)$ is log-concave in v, and $\mathcal{G}(v|a+1/2, c/\rho)$ is log-concave if $a \geq 1/2$. Together,

$$-\log f_0 - \log \mathcal{G}(v|a+1/2, c/\rho) \doteq \frac{1}{2}\log(1+v) - a\log v + \frac{\xi/2}{1+v} - (c/\rho)v,$$

This is convex if $a \geq 1/2$, and the mode of the integrand can be located easily. On the other hand, if $a < 1/2$, this function becomes concave for large enough v. HIER: Work this out.

Note: In `Infer.NET`, such updates are done by using quadrature as well (T. Minka, personal communication). "Note that the posterior for tau can have two modes in some cases" (this is probably the $a < 1/2$ case above).

References

[1] A. Boukouvalas, R. Barillec, and D. Cornford. Gaussian process quantile regression using expectation propagation. In K. Weinberger and A. Globerson, editors, *International Conference on Machine Learning 29*. Omni Press, 2012.

[2] W. Cody. Rational Chebyshev approximations for the error function. *Mathematics of Computation*, 23(107):631–637, 1969.

[3] P. Jylänki, J. Vanhatalo, and A. Vehtari. Robust Gaussian process regression with a Student-t likelihood. *Journal of Machine Learning Research*, 12:3227–3257, 2011.

[4] Q. Liu and D. Pierce. A note on Gauss-Hermite quadrature. *Biometrika*, 81(3):624–629, 1994.

[5] M. Seeger. Low rank updates for the Cholesky decomposition. Technical report, University of California at Berkeley, 2004. See `lapmal.epfl.ch/papers/index.shtml`.

[6] M. Seeger. Bayesian inference and optimal design for the sparse linear model. *Journal of Machine Learning Research*, 9:759–813, 2008.

[7] M. Seeger and H. Nickisch. Compressed sensing and Bayesian experimental design. In A. McCallum, S. Roweis, and R. Silva, editors, *International Conference on Machine Learning 25*, pages 912–919. Omni Press, 2008.

[8] O. Zoeter and T. Heskes. Gaussian quadrature based expectation propagation. In Z. Ghahramani and R. Cowell, editors, *Workshop on Artificial Intelligence and Statistics 10*, 2005.