glm-ie: The Generalised Linear Models Inference & Estimation Toolbox

Hannes Nickisch, Philips Research Europe, Hamburg, Germany

May 18, 2012

Abstract

The glm-ie toolbox contains scalable estimation routines for GLMs (generalised linear models) and SLMs (sparse linear models) as well as an implementation of a scalable convex variational Bayesian inference relaxation. We designed the glm-ie package to be simple, generic and easily expansible. Most of the code is written in Matlab including some MEX files. The code is fully compatible to both Matlab 7.x[1] and GNU Octave 3.3.x[2].

Probabilistic classification, sparse linear modelling and logistic regression are covered in a common algorithmical framework.

Contents

[1] The MathWorks, http://www.mathworks.com/

[2] The Free Software Foundation, http://www.gnu.org/software/octave/

1 Introduction and modelling framework

The `glm-ie` toolbox performs estimation and inference in linear models with unknown parameters $\mathbf{u} \in \mathbb{R}^n$, Gaussian observations

$$\mathbf{y} \quad = \quad \mathbf{X}\mathbf{u} + \boldsymbol{\varepsilon} \in \mathbb{R}^m,\ \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$$

and non-Gaussian potentials $\mathcal{T}_j(s_j)$ on linear projections

$$\mathbf{s} = \mathbf{B}\mathbf{u} \in \mathbb{R}^q$$

leading to a posterior of the form

$$\mathbb{P}(\mathbf{u}|\mathcal{D}) \propto \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{u}, \sigma^2\mathbf{I}) \prod_{j=1}^{q} \mathcal{T}_j(s_j). \tag{1}$$

Note that (sparse) regression as well as classification models fall into the scope of the framework. The algorithm is introduced in the context of MRI sequence optimisation [Seeger, Nickisch, Pohmann, and Schölkopf, 2009, 2010], convexity was proven only later [Nickisch and Seeger, 2009]. For a review of the framework for the general case see Seeger and Nickisch [2008] and the more recent version Seeger and Nickisch [2011].

1.1 MAP estimation

A MAP estimator is the parameter value with highest posterior density

$$\hat{\mathbf{u}}_{\mathrm{MAP}} = \arg\max_{\mathbf{u}} \mathbb{P}(\mathbf{u}|\mathbf{y}).$$

Finding the MAP estimator $\hat{\mathbf{u}}_{\mathrm{MAP}}$ requires the solution of a penalised least squares (PLS) problem

$$\arg\min_{\mathbf{u}} \|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + 2\lambda \cdot \rho(\mathbf{s}),\ \mathbf{s} = \mathbf{B}\mathbf{u},\ \lambda \in \mathbb{R}_+ \tag{2}$$

with penaliser $\rho_{\mathrm{MAP}}(\mathbf{s}) = -\sum_{j=1}^{q} \ln \mathcal{T}_j(s_j)$ and weight $\lambda = \sigma^2$.

1.2 Variational Bayesian inference

The inference algorithm uses the (exact) variational representation

$$\mathcal{T}(s) = \max_{\gamma \geq 0} \exp\left(\beta s - \frac{s^2}{2\gamma} - \frac{h(\gamma)}{2}\right),\ h(\gamma) = \max_{x \geq 0} -\frac{x}{\gamma} - 2g(\sqrt{x}),\ g(s) = \ln \mathcal{T}(s) - \beta s \tag{3}$$

of the potential $\mathcal{T}(s)$ by exploiting its symmetry, positivity and super-Gaussianity, see section 2.4. These bounds can be plugged into the expression for the partition function $Z = \int \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{u}, \sigma^2\mathbf{I}) \prod_{j=1}^{q} \mathcal{T}_j(s_j) \mathrm{d}\mathbf{u}$ to obtain a lower bound thereof

$$Z \geq e^{-\frac{1}{2}h(\gamma)} \int \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{u}, \sigma^2\mathbf{I}) e^{\boldsymbol{\beta}^\top \mathbf{s} - \frac{1}{2}\mathbf{s}\boldsymbol{\Gamma}^{-1}\mathbf{s}} \mathrm{d}\mathbf{u},\ h(\boldsymbol{\gamma}) = \sum_{i=1}^{q} h_i(\gamma_i), \tag{4}$$

where $\mathbf{s} = \mathbf{B}\mathbf{u}$ and $\boldsymbol{\Gamma} = \mathrm{diag}(\boldsymbol{\gamma})$. Evaluating the Gaussian integral yields

$$Z \geq C e^{-\frac{1}{2}\min_\gamma \phi_{VB}(\gamma)} = Z_{VB},\ C = (2\pi)^{\frac{n-m}{2}} \sigma^{-m}$$

where

$$\phi_{VB}(\boldsymbol{\gamma}) = \ln|\mathbf{A}| + h(\boldsymbol{\gamma}) + \min_{\mathbf{u}} R(\mathbf{u}, \boldsymbol{\gamma}),\ R = \frac{1}{\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \mathbf{s}^\top \boldsymbol{\Gamma}^{-1}\mathbf{s} - 2\boldsymbol{\beta}^\top \mathbf{s} \tag{5}$$

and $\mathbf{A} = \mathbf{X}^\top(\sigma^2\mathbf{I})^{-1}\mathbf{X} + \mathbf{B}^\top\boldsymbol{\Gamma}^{-1}\mathbf{B}$.

The Gaussian approximate posterior $\mathbb{P}(\mathbf{u}|\mathcal{D}) \approx \mathbb{Q}(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{V})$ has covariance $\mathbf{V} = \mathbf{A}^{-1}$ with diagonal $\mathbf{z}_\mathbf{u} = \mathrm{dg}(\mathbf{V})$ and mean $\mathbf{m} = \mathbf{A}^{-1}\mathbf{d}$ where $\mathbf{d} = \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{y} + \mathbf{B}^\top\boldsymbol{\beta}$.

Using the concavity of $\boldsymbol{\gamma}^{-1} \mapsto \ln|\mathbf{A}|$, we can upper bound and decouple the term [Wipf and Nagarajan, 2008] by Fenchel duality (between $\omega(\boldsymbol{\gamma})$ and $\omega^*(\mathbf{z})$)

$$\omega(\boldsymbol{\gamma}) = \ln|\mathbf{A}| = \min_{\mathbf{z}} \mathbf{z}^\top\boldsymbol{\gamma}^{-1} - \omega^*(\mathbf{z}),\ \mathbf{z} > \mathbf{0},\ \text{where } \mathbf{z}_* = \arg\min_{\mathbf{z}} \mathbf{z}^\top\boldsymbol{\gamma}^{-1} - \omega^*(\mathbf{z}) = \mathrm{dg}(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top) \tag{6}$$

to obtain the (jointly convex in $(\mathbf{u}, \gamma)$ for log-concave potentials) variational criterion

$$\begin{aligned}\phi_{VB}(\gamma, \mathbf{u}, \mathbf{z}) &= \mathbf{z}^\top \gamma^{-1} - \omega^*(\mathbf{z}) + h(\gamma) + \frac{1}{\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \mathbf{s}^\top \boldsymbol{\Gamma}^{-1}\mathbf{s} - 2\boldsymbol{\beta}^\top \mathbf{s} \\ &= \frac{1}{\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \sum_{i=1}^{q} h_i(s_i, \gamma_i) - \omega^*(\mathbf{z}),\ h_i(s_i, \gamma_i) = \frac{z_i + s_i^2}{\gamma_i} + h_i(\gamma_i) - 2\beta_i s_i\end{aligned} \tag{7}$$

which decouples into scalar problems w.r.t. γ and is of PLS structure w.r.t. $\mathbf{u}$. Note that $\phi_{VB}(\gamma) = \min_{\mathbf{u},\mathbf{z}>\mathbf{0}} \phi_{VB}(\gamma, \mathbf{u}, \mathbf{z})$. For log-concave potentials $\mathcal{T}(s)$, we can do univariate minimisations in γ_i in closed form

$$h^*(s) = \frac{1}{2}\min_{\gamma \geq 0} h(s, \gamma) = \min_{\gamma \geq 0} \frac{1}{2}\left[\frac{z + s^2}{\gamma} + h(\gamma)\right] - \beta s = \beta \cdot (\varsigma - s) - \ln \mathcal{T}(\varsigma),\ \varsigma = \text{sign}(s) \cdot \sqrt{s^2 + z} \tag{8}$$

by matching the variational representation of $\mathcal{T}(s)$ from equation 3 $-2g(\sqrt{x}) = \min_{\gamma \geq 0} x/\gamma + h(\gamma)$ for $x = z + s^2$ where we have dropped the indices i. We see that

1. for $z = 0$, we have $h^*(s) = -\ln \mathcal{T}(s)$ and
2. for $s \to \pm 0$, $h^*(s)$ is continuous, due to the symmetry $f_\beta(s) = f_\beta(-s)$ of $f_\beta(s) = \mathcal{T}(s)e^{-\beta s}$ from section 2.4 yields $h^*(-s) = h^*(s) + 2\beta s$.

Hence, the γ dependence can be dropped from the variational criterion by performing the minimisation w.r.t. γ analytically

$$\begin{aligned}\phi_{VB}(\mathbf{u}, \mathbf{z}) &= \min_{\gamma} \phi_{VB}(\gamma, \mathbf{u}, \mathbf{z}) = \frac{1}{\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + 2 \cdot h^*(\mathbf{B}\mathbf{u}) - \omega^*(\mathbf{z}) \\ &\quad \text{where } h^*(\mathbf{s}) = \boldsymbol{\beta}^\top(\boldsymbol{\varsigma} - \mathbf{s}) - \ln \mathcal{T}(\boldsymbol{\varsigma}),\ \boldsymbol{\varsigma} = \text{sign}(\mathbf{s}) \odot \sqrt{\mathbf{s}^2 + \mathbf{z}}.\end{aligned} \tag{9}$$

The (non-zero) sign function is understood as $\text{sign}(t) = t/|t|$ with $\text{sign}(0) = 1$. The optimal value $\gamma_* = \arg\min_\gamma \phi(\gamma, \mathbf{u}, \mathbf{z})$ obtained as

$$\begin{aligned}\gamma_*^{-1} &= -2\frac{\partial g(\varsigma)}{\partial \varsigma^2},\ \varsigma = \text{sign}(\mathbf{s}) \odot \sqrt{\mathbf{s}^2 + \mathbf{z}}.,\ g(\mathbf{s}) = \ln \mathcal{T}(\mathbf{s}) - \boldsymbol{\beta}^\top \mathbf{s} \\ &= \frac{\boldsymbol{\beta} - [\ln \mathcal{T}]'(\varsigma)}{\varsigma}\end{aligned}$$

for $\boldsymbol{\tau}$-scaled potentials $\mathcal{T}_i(\tau_i s_i)$, we obtain the expression $\gamma_*^{-1} = \frac{\boldsymbol{\tau} \odot \boldsymbol{\beta} - [\ln \mathcal{T}]'(\boldsymbol{\tau} \odot \varsigma)}{\varsigma}$.

1.2.1 Double loop algorithm

Our double loop algorithm (see section 2.5) minimises $\phi_{VB}(\mathbf{u}, \mathbf{z})$ by iterating between minimisation in one variable while keeping the other one fixed.

1. Outer loop: $\mathbf{z}_* = \arg\min_{\mathbf{z}} \phi_{VB}(\mathbf{u}, \mathbf{z}) = \text{dg}(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top) = \mathbb{V}[\mathbf{s}|\mathcal{D}]$
2. Inner loop: $\mathbf{u}_* = \arg\min_{\mathbf{u}} \phi_{VB}(\mathbf{u}, \mathbf{z}) = \arg\min_{\mathbf{u}} \frac{1}{\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + 2 \cdot h^*(\mathbf{B}\mathbf{u}) = \mathbb{E}[\mathbf{u}|\mathcal{D}]$

The outer loop is a variance estimation problem and the inner loop is a penalised least squares or MAP estimation problem (equation 2) with penaliser $\rho_{\text{VB}}(\mathbf{s}) = h^*(\mathbf{s})$ and weight $\lambda = \sigma^2$.

1.2.2 Nonlinear or group potentials

All the maths so far deal with linear potentials i.e. scaled super-Gaussian potentials acting on scaled *linear* projections: $\mathcal{T}_i(\mathbf{u}) = \mathcal{T}_i(\tau \cdot \mathbf{b}_i^\top \mathbf{u}) = \mathcal{T}_i(\tau \cdot s_i)$. What happens if we were to allow *non-linear* functions? If the integral of equation 4

$$I = \int \exp\left(-\frac{1}{2}(\underbrace{\mathbf{s}^\top \boldsymbol{\Gamma}^{-1}\mathbf{s} - 2\boldsymbol{\beta}^\top \mathbf{s}}_{\delta(\mathbf{u})}) - \frac{1}{2\sigma^2}\|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2\right) \text{d}\mathbf{u}$$

is Gaussian, we can solve it. This is exactly the case if $\delta(\mathbf{u}) = \mathbf{s}^\top \boldsymbol{\Gamma}^{-1}\mathbf{s} - 2\boldsymbol{\beta}^\top \mathbf{s}$ is a quadratic function in $\mathbf{u}$. There are exactly two possibilities:

1. A linear map $\mathbf{u} \mapsto \mathbf{s}$ i.e. $\mathbf{s} = \mathbf{B}\mathbf{u}$ naturally yields a quadratic $\delta(\mathbf{u}) = \mathbf{u}^\top \mathbf{B}^\top \boldsymbol{\Gamma}^{-1} \mathbf{B}\mathbf{u} - 2\boldsymbol{\beta}^\top \mathbf{B}\mathbf{u}$.

2. Using $\beta_i = 0$ and Euclidean norms $\tilde{s}_i = \sqrt{\mathbf{s}^\top \mathbf{G}_i \mathbf{s}}$ with positive definite weighting matrices $\mathbf{G}_i \in \mathbb{R}^{q \times q}$, $i = 1..\tilde{q}$. We obtain the quadratic function $\tilde{\delta}(\mathbf{u}) = \sum_{i=1}^{\tilde{q}} \tilde{s}_i^2 / \tilde{\gamma}_i = \mathbf{u}^\top \mathbf{B}^\top \left(\sum_{i=1}^{\tilde{q}} \mathbf{G}_i / \tilde{\gamma}_i \right) \mathbf{B}\mathbf{u}$ where $\tilde{\boldsymbol{\gamma}} \in \mathbb{R}_+^{\tilde{q}}$.

In order to work with the second possibility, we sacrifice some modelling power in order to gain computational and notational efficiency by working with diagonal matrices $\mathbf{G}_i = \mathrm{dg}(\mathbf{g}_i)$. We use a single grouping matrix $\mathbf{G} \in \mathbb{R}_+^{\tilde{q} \times q}$ to accommodate the $\tilde{q}$ row vectors $\mathbf{g}_i \in \mathbb{R}_+^q$. We obtain $\tilde{\delta}(\mathbf{u}) = \mathbf{s}^\top \mathrm{dg}\left(\mathbf{G}^\top \tilde{\boldsymbol{\gamma}}\right) \mathbf{s}$ which allows to write $\mathbf{G}^\top \tilde{\boldsymbol{\gamma}} = \boldsymbol{\gamma}$ in the following.

In the outer loop, we compute $\tilde{\mathbf{z}} = \mathbf{G}\mathbf{z}$, where $\mathbf{z} = \mathrm{dg}(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top)$ and $\mathbf{A} = \mathbf{X}^\top \mathbf{X}/\sigma^2 + \mathbf{B}^\top \boldsymbol{\Gamma}^{-1} \mathbf{B}$.

The inner loop (equation 9) is more complicated since the (even due to $\beta_i = 0$) penalty $h_i^*(|s_i|)$ has to be replaced by $\tilde{h}_i^*(\mathbf{s}) = h_i^*(|\tilde{s}_i|) = h_i^*\left(\sqrt{\mathbf{s}^\top (\mathbf{g}_i \odot \mathbf{s})}\right)$ which can equivalently be expressed as

$$\tilde{h}^*(\mathbf{s}) = -\ln \mathcal{T}\left(\sqrt{\tilde{\mathbf{s}}^2 + \tilde{\mathbf{z}}}\right) = -\ln \mathcal{T}(\boldsymbol{\varsigma}),\ \boldsymbol{\varsigma} = \sqrt{\mathbf{G}(\mathbf{s}^2 + \mathbf{z})}.$$

The standard inner loop objective $\phi(\mathbf{u}) = \frac{1}{2\sigma^2} \|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + h^*(\mathbf{s})$ has gradient $\frac{\partial \phi(\mathbf{u})}{\partial \mathbf{u}} = \sigma^{-2} \mathbf{X}^\top (\mathbf{X}\mathbf{u} - \mathbf{y}) + \mathbf{B}^\top \mathbf{h}'_*(\mathbf{s})$ and Hessian $\frac{\partial^2 \phi(\mathbf{u})}{\partial \mathbf{u} \partial \mathbf{u}^\top} = \sigma^{-2} \mathbf{X}^\top \mathbf{X} + \mathbf{B}^\top \mathbf{H}''_*(\mathbf{s}) \mathbf{B}$, where $h^*(\mathbf{s}) = \sum_{i=1}^q h_i^*(s_i)$, $\mathbf{s} = \mathbf{B}\mathbf{u}$. For nonlinear potentials $\mathcal{T}$, the terms $\tilde{\mathbf{h}}'_*(\mathbf{s}) = \frac{\partial}{\partial \mathbf{s}} \tilde{h}^*(\mathbf{s})$ and $\tilde{\mathbf{H}}''_*(\mathbf{s}) = \frac{\partial^2}{\partial \mathbf{s} \partial \mathbf{s}^\top} \tilde{h}^*(\mathbf{s})$ become more complicated i.e. $\tilde{\mathbf{H}}''_*$ is no longer a diagonal matrix. We have

$$\begin{aligned}
\tilde{\mathbf{h}}'_*(\mathbf{s}) &= \sum_{i=1}^{\tilde{q}} \frac{\partial}{\partial \mathbf{s}} h_i^*(\tilde{\varsigma}_i) = \sum_{i=1}^{\tilde{q}} \frac{h_i'^*(\tilde{\varsigma}_i)}{\tilde{\varsigma}_i} \mathbf{g}_i \odot \mathbf{s} = \left(\mathbf{G}^\top \left[\mathbf{h}'_*(\tilde{\boldsymbol{\varsigma}}) \odot \tilde{\boldsymbol{\varsigma}}^{-1} \right] \right) \odot \mathbf{s},\ \tilde{\boldsymbol{\varsigma}}^2 = \mathbf{G}(\mathbf{s}^2 + \mathbf{z}) \\
&= \mathbf{S} \underbrace{\mathbf{G}^\top \tilde{\boldsymbol{\Omega}}^{-1} \mathbf{h}'_*(\tilde{\boldsymbol{\varsigma}})}_{\mathbf{v}},\ \mathbf{S} = \mathrm{dg}(\mathbf{s}),\ \tilde{\boldsymbol{\Omega}} = \mathrm{dg}(\tilde{\boldsymbol{\varsigma}}) \text{ and} \\
\tilde{\mathbf{H}}''_*(\mathbf{s}) &= \frac{\partial}{\partial \mathbf{s}^\top} \mathbf{S}\mathbf{G}^\top \tilde{\boldsymbol{\Omega}}^{-1} \mathbf{h}'_*(\tilde{\boldsymbol{\varsigma}}) \\
&= \mathbf{S}\mathbf{G}^\top \mathrm{dg}(\mathbf{w}) \mathbf{G}\mathbf{S} + \mathrm{dg}(\mathbf{v}),\ \mathbf{w} = \mathbf{h}''_*(\tilde{\boldsymbol{\varsigma}}) \odot \tilde{\boldsymbol{\varsigma}}^{-2} - \mathbf{h}'_*(\tilde{\boldsymbol{\varsigma}}) \odot \tilde{\boldsymbol{\varsigma}}^{-3}.
\end{aligned}$$

The optimal value $\boldsymbol{\gamma}_* = \mathbf{G}^\top \tilde{\boldsymbol{\gamma}}_*$ is computed as

$$\tilde{\gamma}_*^{-1} = -2 \frac{\partial \ln \mathcal{T}(\tilde{\varsigma})}{\partial \tilde{\varsigma}^2} = -\frac{[\ln \mathcal{T}]'(\tilde{\varsigma})}{\tilde{\varsigma}},\ \tilde{\varsigma} = \sqrt{\mathbf{G}(\mathbf{s}^2 + \mathbf{z})}.$$

1.3 Mean field inference

Instead of aiming for a full Gaussian $\mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{V})$ as posterior approximation, the mean field approach [Miskin and MacKay, 2000] uses a factorial distribution $\mathcal{N}(\mathbf{u}|\mathbf{m}, \mathrm{dg}(\mathbf{v}))$ instead.

1.3.1 Double loop algorithm

In fact, the variational bounding inference algorithm (section 1.2) can be used for mean field with a modified outer loop update

$$\mathbf{v}^{-1} \leftarrow \mathrm{dg}(\mathbf{A}),\ \mathbf{z} \leftarrow \mathrm{dg}(\mathbf{B}\mathrm{dg}(\mathbf{v})\mathbf{B}^\top),$$

which can be computed efficiently and exactly for sparse matrices, Fourier matrices and convolution matrices. Note that $(\mathbf{A} \odot \mathbf{A})\mathbf{x} = \mathrm{dg}(\mathbf{A}\mathrm{dg}(\mathbf{x})\mathbf{A}^\top)$.

1.4 Expectation propagation inference

Expectation propagation (EP) is a more general and often more accurate inference algorithm that does not require the potentials to be super-Gaussian. The approximating family of distributions $\mathbb{Q}(\mathbf{u})$ is the same as in variational Bayes but $\boldsymbol{\beta}$ is treated as an additional parameter. The criterion i.e. the EP free energy

$$\phi_{EP}(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \ln |\mathbf{A}| + h_{EP}(\boldsymbol{\gamma}, \boldsymbol{\beta}) + \min_{\mathbf{u}} R(\mathbf{u}, \boldsymbol{\gamma}, \boldsymbol{\beta}),\ R = \frac{1}{\sigma^2} \|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \mathbf{s}^\top \boldsymbol{\Gamma}^{-1} \mathbf{s} - 2\boldsymbol{\beta}^\top \mathbf{s}$$

where $\mathbf{A} = \mathbf{X}^\top \mathbf{X}/\sigma^2 + \mathbf{B}^\top \boldsymbol{\Gamma}^{-1}\mathbf{B}$ is similar to the variational Bayesian criterion $\phi_{VB}(\boldsymbol{\gamma})$ the difference being $h_{EP}(\boldsymbol{\gamma}, \boldsymbol{\beta})$.

Let $\mathbf{m} = \mathbb{E}_{\mathbb{Q}}[\mathbf{u}|\mathcal{D}]$ and $\mathbf{V} = \mathbb{V}_{\mathbb{Q}}[\mathbf{s}|\mathcal{D}]$, then the marginal distribution $\mathbb{Q}_j(s_j)$ is given by $\mathcal{N}(s_j|\mu_j, \rho_j)$ where $\boldsymbol{\mu} = \mathbf{Bm}$ are the marginal means and $\boldsymbol{\rho} = \mathrm{dg}(\mathbf{BVB}^\top)$ are the marginal variances. The cavity marginals $\mathbb{Q}_{\neg j}(s_j) \propto \mathbb{Q}_j(s_j) e^{-\eta(s_j\beta_j - s_j^2/2\gamma_j)} \propto \mathcal{N}(s_j|\mu_{\neg j}, \rho_{\neg j})$ and tilted marginals $\hat{\mathbb{P}}(s_j) \propto \mathbb{Q}_{\neg j}(s_j)\mathcal{T}_j(s_j)^\eta$ with fractional parameter $\eta \in (0, 1]$ can be used to define

$$h_j(\gamma_j, \beta_j) \quad = \quad -\frac{2}{\eta}\left(\ln \mathbb{E}_{\mathbb{Q}_{\neg j}}[\mathcal{T}_j(s_j)^\eta] - \ln \mathbb{E}_{\mathbb{Q}_{\neg j}}[e^{\eta(s_j\beta_j - s_j^2/2\gamma_j)}]\right), \; \mathbb{E}_{\mathbb{Q}_{\neg j}}[f(s_j)] = \int f(s_j)\mathbb{Q}_{\neg j}(s_j)\mathrm{d}s_j.$$

The EP algorithm finds a stationary point (a saddlepoint in general) of the EP free energy by iterative local moment matching i.e. setting (γ_j, β_j) such that $\hat{\mathbb{P}}(s_j) \propto \mathbb{Q}_{\neg j}(s_j)\mathcal{T}_j(s_j)^\eta$ and $\mathbb{Q}_j(s_j) \propto \mathbb{Q}_{\neg j}(s_j) e^{\eta(s_j\beta_j - s_j^2/2\gamma_j)}$ have the same first two moments (r_j, z_j).

1.4.1 Parameter update details

Define the precision $\pi_j = \gamma_j^{-1}$, then the EP updates have the form

$$\begin{aligned} \pi_j &\leftarrow (1-\eta)\pi_j - \frac{d_2}{1 + d_2\rho_{\neg j}}, \; d_2 = \frac{\partial^2 \ln \hat{Z}_j}{\partial \mu_{\neg j}^2} \\ \beta_j &\leftarrow (1-\eta)\beta_j + \frac{d_1 - d_2\mu_{\neg j}}{1 + d_2\rho_{\neg j}}, \; d_1 = \frac{\partial \ln \hat{Z}_j}{\partial \mu_{\neg j}} \end{aligned}$$

where $\rho_{\neg j} = \frac{\rho_j}{1-\eta\pi_j\rho_j}$, $\mu_{\neg j} = \frac{\mu_j - \eta\beta_j\rho_j}{1-\eta\pi_j\rho_j}$ and $\hat{Z}_j = \int \mathbb{Q}_{\neg j}(s_j)\mathcal{T}(s_j)^\eta \mathrm{d}s_j$.

1.4.2 Double loop algorithm

In our implementation, we employ parallel EP updates [van Gerven et al., 2010] that is we update all parameters $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ jointly. The outer loop computes $\boldsymbol{\rho} = \mathbf{z} = \mathrm{dg}(\mathbf{BA}^{-1}\mathbf{B}^\top)$ as in VB, section 1.2. The inner loop iterates between doing parallel EP updates and recomputation of $\boldsymbol{\mu} = \mathbf{Bm}$. Unfortunately, EP seems to be less robust against imperfect computation of $\mathbf{z}$. Whereas VB can deal with underestimated marginal variances $\mathbf{0} < \hat{\mathbf{z}} \ll \mathbf{z}$ naturally, EP demands exact variance computations. This considerably limits the scope of EP inference.

1.5 Package organisation

Besides the illustrating documentation and example in `doc/` and the double loop inference engine (section 2.5) , the package naturally splits into four functional objects like matrix operators, PLS solvers, penalty and potential functions.

Directory	Content
`doc/`	documentation and demo code
`inf/`	double loop inference code, section 2.5
`mat/`	matrix classes, section 2.1
`pls/`	penalised least squares code, section 2.2
`pen/`	penalty functions $\rho(s)$, section 2.3
`pot/`	potential functions $\mathcal{T}(s)$, section 2.4

If you wish to add more functional objects, the only thing you have to do is to implement the interface as detailed in sections 2.1, 2.2, 2.3 and 2.4.

2 Code

In the following, we detail the function objects of the `glm-ie` toolbox: matrix operators (section 2.1), PLS solvers (section 2.2), penalty functions (section 2.3) and potential functions (section 2.4) needed to successfully operate the inference engine (section 2.5).

2.1 Matrix operators in `mat/`

For the `glm-ie` toolbox, a matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ is completely specified by its MVMs (matrix vector multiplications) $\mathbf{A}\mathbf{x} \in \mathbb{C}^m$ and $\mathbf{A}^\top \mathbf{y} \in \mathbb{C}^n$. We use of Matlab's object oriented programming facilities to provide a basic (lazily evaluated) matrix class that is completely generic and can be fully instantiated by defining an MVM. The design of our matrix class allows to use exactly the same code for dense and sparse matrices as well as matrices implicitly specified through their MVM. A user defines the MVM and the matrix class makes sure that an instance inherits all its properties. In the following, we will describe the general matrix class and its implementations.

In the `glm-ie` toolbox, a matrix $\mathbf{A}$ has to be able to deal with complex numbers. However, the potentials for the probabilistic model act on real-valued variables s_i only. Therefore, complex vectors have to be embedded into real vector spaces. We represent the complex vector $\mathbf{v} \in \mathbb{C}^n$ by the real vector $\mathbf{w} = [\Re z_1, \Im z_1, \Re z_2, .., \Re z_n, \Im z_n]^\top \in \mathbb{R}^{2n}$ that has twice the number of components and contains real and imaginary parts in an interleaved way. Conversions between $\mathbf{v}$ and $\mathbf{w}$ can be done by the two auxiliary functions `mat/re2cx.m` and `mat/cx2re.m` such that $\mathbf{w} = \texttt{cx2re}(\mathbf{v})$ and $\mathbf{v} = \texttt{re2cx}(\mathbf{w})$. Similarly if the complex matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ is represented by $\begin{bmatrix} \Re\mathbf{A} & -\Im\mathbf{A} \\ \Im\mathbf{A} & \Re\mathbf{A} \end{bmatrix} \in \mathbb{R}^{2m \times 2n}$, then complex MVMs can be done by code written for real variables.

2.1.1 The matrix class `mat`

The philosophy behind the matrix class is that all properties of a matrix can be expressed in terms of MVMs. A general MVM has a computational complexity of $\mathcal{O}(m \cdot n)$. Since this computational load is prohibitive for m and n in the order of number of pixels of an image, our focus is on structured matrices reducing the cost to $\mathcal{O}(n \cdot \log n)$ at most. The functionality of our base class named `mat` is stored in the directory `mat/@mat/`.

Our matrix class offers simple "infrastructure" functionality to determine the size, display information and printing. The following functions are available:

1. `disp.m` displays some information about the size and the class of $\mathbf{A}$,
2. `full.m` returns the full matrix $\mathbf{A}$,
3. `imagesc.m` displays $\mathbf{A}$ as an image,
4. `isempty.m` indicates whether there is no entry in $\mathbf{A}$,
5. `numel.m` counts the number of elements in $\mathbf{A}$,
6. `size.m` returns the size of the matrix as stored in the struct field `A.sz`,
7. `real.m`, `imag.m`, `abs.m`, `angle.m` return full matrices and
8. `end.m` is needed for indexing.

Furthermore, matrix valued functions of matrices such as addition, concatenation, Kronecker product, multiplication, replication, scaling, subreferencing or transposition allow to create new matrices from existing ones. Matlab and Octave's matrix objects require several functions to be specified for a matrix object $\mathbf{A}$ (named `A` in the code) of type `mat` that are called whenever the interpreter encounters expressions such as `A*B` for $\mathbf{AB}$ or the like. All matrix functions in the `glm-ie` toolbox are listed in the following table:

Mathematical operation		Matlab expression	Class function
MVM	$\mathbf{Ax}$, $\mathbf{x}^\top\mathbf{A}$	`A*x`, `x'*A`	`mvm.m`
MVM	$(\mathbf{A}\odot\mathbf{A})\mathbf{x}$, $\mathbf{x}^\top(\mathbf{A}\odot\mathbf{A})$	`mvmAA(A,x)`	`mvmAA.m`
concatenation	$[\mathbf{B},\mathbf{C}]$	`[A,B]`	`horzcat.m`
concatenation	$[\mathbf{B};\mathbf{C}]$	`[A;B]`	`vertcat.m`
difference	$\mathbf{A}-\mathbf{B}$	`A-B`	`minus.m`
tensor product	$\mathbf{A}\otimes\mathbf{B}$	`kron(A,B)`	`kron.m`
multiplication	$\mathbf{AB}$	`A*B`	`mtimes.m`
negation	$-\mathbf{A}$	`-A`	`uminus.m`
replication	$[\mathbf{A},\mathbf{A}]$	`repmat(A,[1,2])`	`repmat.m`
scaling	$\alpha\mathbf{A}$, $\mathbf{A}\alpha$	`al*A`, `A*al`	`mtimes.m`
sum	$\mathbf{A}+\mathbf{B}$	`A+B`	`plus.m`
subreferencing	$\mathbf{A}_{:,j}$, $\mathrm{vec}(\mathbf{A})$	`A(:,j)`, `A(:)`	`subsref.m`
transposition	$\mathbf{A}^\top$	`A'`	`ctranspose.m`
vectorisation	$\mathrm{vec}(\mathbf{A})$	`vec(A)`	`vec.m`

Most importantly, computations are only done when an MVM is called; only then the tree of composition is traversed and all the atomic MVMs are executed and combined – the computational load is entirely concentrated in the functions `mvm.m` that have to be specified by all matrix operators implementing the matrix class. The auxiliary functions `type.m` and `args.m` can be used to get access to the composition tree.

Complex numbers can be handled in four different ways by setting the `complex` argument in the constructor `mat/@mat/mat.m`. Both input and output vectors can separately be specified to be composed of complex numbers $\Re z + \imath\Im z$ pairs of real numbers $[\Re z; \Im z]$ or just a real numbers $\Re z$.

2.1.2 Implementations

The `glm-ie` toolbox currently includes 13 matrix objects $\mathbf{A}$ derived from `mat/@mat` whose code is located in the directories `mat/@mat<NAME>/` and its subdirectories. Every matrix object needs two things

1. `mat/@mat<NAME>/mat<NAME>.m` is the constructor for the matrix object and
2. `mat/@mat<NAME>/mvm.m` implements the MVM depending on the transpose flag `ctransp`.

All the rest is inherited from from `mat/@mat`. We include the diagonal matrix class `mat/@matDiag` as a very simple example serving as a starting point for additional implementations.

`<NAME>`	$\mathbf{A}$	Meaning	Formal $\mathbf{Ax}\equiv$	$\mathcal{O}(n)=$
`Conv2`	$\mathbf{K}$	Convolution with kernel $\mathbf{k}$ (adapted from Michael Hirsch)	$\mathbf{k}\star\mathbf{x}$	n
`Diag`	$\mathbf{D}$	Diagonal matrix	$\mathbf{d}\odot\mathbf{x}$	n
`FD2`	$\mathbf{D}$	Finite derivatives in both directions	$[x_{i+1}-x_i]_i$	n
`FFTN`	$\mathbf{F}$	Fourier matrix in N dimensions	$\mathbf{F}$	$n\cdot\ln n$
`FFTNmask`	$\mathbf{F}_\mathbf{M}$	Partial Fourier, single points	$(\mathbf{Fx})_\mathbf{M}$ subset	$n\cdot\ln n$
`FFT2line`	$\mathbf{F}_\mathbf{l}$	Partial Fourier, single rows $\mathbf{l}\in\mathbb{N}_+^m$	$(\mathbf{Fx})_\mathbf{l}$ subset	$n\cdot\ln n$
`FFT2nu`	$\mathbf{F}_\mathbf{k}$	Non-uniform Fourier at $\mathbf{k}\in\mathbb{C}^m$ (based on `nufft`)	$\approx\mathbf{P}_\mathbf{k}\mathbf{F}\mathrm{dg}(\mathbf{d})$	$n\cdot\ln n$
`Wav`	$\mathbf{W}$	Fast wavelet transform (using `fwtn`)	$\mathbf{Wx}$	n

2.1.3 Example matrix

Implementation of a custom matrix is very simple. Suppose, we wanted to have a matrix object that imitates the behavior of the identity matrix. Let's call the object `matEye` for the time being. Then the following steps are sufficient:

- 1) create a new directory `mat/@matEye`
- 2) create a file `mat/@matEye/matEye.m`

```
% Identity matrix example
function A = matEye(n,m, par, complex)
  sz = [n,m];                                    % size of the matrix
```

```
  if nargin<3, A.par = ''; else A.par = par; end         % additional parameters
  if nargin<4, complex = 1; end
  if numel(complex)==1, complex = complex*[1,1]; end
  A = class(A,mfilename,mat(sz,complex));
```

- 3) create a file `mat/@matEye/mvm.m`

```
% Identity matrix example
function y = mvm(A,x,ctransp)
  sz = size(A);                                                    % size
  p = A.par;                                                % additional parameter
  if ctransp                                                        % transposed
    fprintf('We did an MVM with eye(%d,%d)'' par=%s.\n',sz,p)
    y = x;
  else                                                          % not transposed
    fprintf('We did an MVM with eye(%d,%d) par=%s.\n',sz,p)
    y = x;
  end
```

- 4) use it on the command prompt
  - `>> n = 100; m = 50;`
  - `>> I = matEye(n,m,'example');`
  - `>> x = randn(m,1);`
  - `>> y = I*x;`

The example is part of the code release.

Note on Octave compatibility

There is an issue with the way Octave[3] evaluates expressions of the form `x=A'*y`; namely that it will terminate with an `T& Array<T>::checkelem (2): range error` because the transpose seems to be evaluated in a lazy fashion. Our workaround in the `glm-ie` toolbox is to use expressions `x=[A']*y`; instead to force the creation of a new object being the transpose of the original one. Both Matlab and Octave are smart enough to avoid creating full copies of `A` – leading to essentially no memory overhead. In summary, you have to use Octave 3.3.x in order to fully enjoy the `glm-ie` toolbox.

[3] Until version 3.2.4, fixed in 3.3.x.

2.2 Penalised least squares solvers in `pls/`

In the `glm-ie` toolbox, PLS estimation problems (equation 2) with tradeoff parameter $\lambda > 0$

$$\arg\min_{\mathbf{u}} \frac{1}{2} \|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \lambda \cdot \rho(\mathbf{B}\mathbf{u}; \boldsymbol{\psi})$$

are encountered both in MAP estimation (section 1.1) and in the inner loop (section 1.2.1) of our double loop variational inference method.

2.2.1 Interface

We provide a generic interface in `plsSolvers.m` to be able to use a variety of different PLS solvers. In order to run the methods, you need to provide a starting value $\mathbf{u}_0$, the matrices $\mathbf{X}$ and $\mathbf{B}$, the measurement vector $\mathbf{y}$, the optimisation parameters stored in the struct `opt`, the weight λ and the penaliser $\rho(\mathbf{s}; \boldsymbol{\psi})$ with additional parameters $\boldsymbol{\psi}$ stored in the cell array `varargin`.

```
% A PLS (penalised least squares) solver is a program solving the
% minimisation problem
%   phi(u) = 1/lambda * ||X*u-y||_2^2 + 2*sum( pen(s) ), s = B*u,
% where pen(s) is a penalty function.
%
% [u,phi] = pls<NAME>(u0,X,y,B,opt,lam,pen,varargin)
%
% INPUT
% =====
%    u0  [nx1]  initial vector
%    X   [mxn]  matrix or operator
%    y   [mx1]  vector
%    B   [qxn]  matrix or operator
%    opt.       optimisation parameters
%      nMVM     maximal number of steps = matrix vector multiplications ...
%                           with A=X'*X/lam+B'*D*B, D diagonal,   [default 100]
%      output   flag saying whether something is printed         [default false]
%               the function will show the current iteration number, the actual
%               function value phi and the current length of the step
%               in the format:                 13, phi=8.6063e-01;  du=2.325e-04
%    lam [1x1]  (positive) weight of the penaliser, can be Inf
%    pen        penaliser function handle or function name
%                                       [p,dp,d2p] = feval(pen,Bu,varargin{:})
%    varargin   additional parameters for pen
%
% OUTPUT
% ======
%    u   [nx1]  optimal solution
%    phi [1x1]  optimal function value
%
% Currently, we have implemented in pls/pls<NAME>.m
%   LBFGS: LIMITED memory BROYDEN-FLETCHER-GOLDFARB-SHANNO
%               quasi Newton or variable metric method
%   TN:    TRUNCATED NEWTON
%               optimisation with CG approximated Newton steps
%   CGBT:  CONJUGATE GRADIENTS with BACKTRACKING line search
%               optimisation using the Armijo rule
%   CG:    CONJUGATE GRADIENTS
%               optimisation with CG code minimize.m by Carl E. Rasmussen
%   SB:    SPLIT BREGMAN
%               optimisation using an augmented Lagrangian approach
%   BB:    BARZILAI/BORWEIN
%               stepsize adjusted gradient method without descent guarantee
%
% Examples:
% >> lam=1; plsLBFGS(u,X,y,B,opt,lam,'penQuad')
% >> lam=1; tau=2; z=1.3; plsCG(u,X,y,B,opt,lam,'penVB','potLaplace',tau,z)
%
%   See also PENFUNCTIONS.M, POTFUNCTIONS.M.
```

```
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2011 October 20
```

2.2.2 Implementations

All currently implemented solvers can be found at `pls/pls<NAME>.m`.

<NAME>	Meaning
CG	Conjugate gradients (CG) using gradient info in the line searches.
CGBT	CG with backtracking line search using Armijo's rule.
LBFGS	Limited memory Broyden–Fletcher–Goldfarb–Shanno quasi Newton.
SB	Bregman Splitting; an augmented Lagrangian approach.
BB	Barzilai-Borwein first order two point stepsize rule.
TN	Truncated Newton with CG approximated Newton direction.

2.2.3 Auxiliary routines

- In `pls/plsCG.m`, the conjugate gradient optimiser `minimize.m` by Carl E. Rasmussen is called with the objective `pls/phi.m`.
- A second conjugate gradient solver `pls/plsCGBT.m` inspired by the `fnlCg.m` code is available; it also makes calls to `pls/phi.m`.
- With `pls/plsLBFGS.m`, we also have an interface to the powerful general purpose optimiser L-BFGS-B written in Fortran (see section 3). Its code can be found in `pls/lbfgsb/`; the corresponding binaries along with installation instructions are contained in the files `pls/lbfgsb.{m,mex*}`, respectively. We have another Matlab wrapper contained in `pls/minimize_lbfgsb_*.m` having exactly the same interface as `minimize.m`.
- The Bregman splitting approach [Combettes and Pesquet, 2010, Goldstein and Osher, 2009] implemented in `pls/plsSB.m` is a very fast method whenever the matrices $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{B}^\top\mathbf{B}$ are diagonal in the Fourier space (see the function `mat/@mat<NAME>/diagFAtAFt.m`) i.e. solving a linear system with them takes only $\mathcal{O}(n \cdot \ln n)$ time due to the fast Fourier transform. The code is inspired by `mrics.m`. Another important ingredient for Bregman splitting is the evaluation of scalar proximity operators $\mathrm{prox}_\lambda(r) = \arg\min_s \frac{1}{2\lambda}(s-r)^2 + \rho(s)$ for a given penalty function $\rho(s)$. In `pls/prox.m`, we have implemented a generic Newton algorithm that handles any differentiable penalty function and analytic solutions where applicable. We have analytic solutions for `penAbs.m`, `penQuad.m`, `penZero.m` and `penVB.m`/`potLaplace.m`, see sections 2.3 and 2.4. The latter requires the solution of quartic equations which can be obtained by `pls/solve_quartic.m` and `pls/solve_cubic.m`.
- A simple first order solver based on a two-point step size rule [Barzilai and Borwein, 1988] has very good empirical performance in high-dimensional problems at low computational cost. The algorithm implemented in `pls/plsBB.m` is not guaranteed to decrease the objective in every step but performs very well on large ℓ_1 type problems.
- Finally, we have a truncated Newton procedure `pls/plsTN.m` running. This approach is also known as Iteratively Reweighted Least Squares (IRLS). Here, the line search along the Newton search direction is done by Brent's golden section search `pls/brentmin.m`. The computation of the Newton search direction itself requires the (approximate) solution of a linear system. We have two rather general linear system solvers allowing to find a vector $\mathbf{c}$ such that $\mathbf{Ac} = \mathbf{b}$ where $\mathbf{A} = \mathbf{X}^\top\mathbf{RX} + \mathbf{B}^\top\mathbf{PB}$:
  - Linear conjugate gradients `pls/linsolve_lcg.m` and
  - Full inversion using the Woodbury identity `pls/linsolve_woodbury.m`, see section 2.5 and `inf/diaginv_woodbury.m` therein.

2.3 Penalty functions $\rho(s)$ in `pen/`

Penalty functions are used to shape the PLS problem (equation 2); continuous differentiability is our minimum requirement. We do not require convexity of $\rho(s)$, but the optimisation becomes much simpler since convexity of $\rho(s)$ implies convexity of the entire PLS problem.

2.3.1 Interface

Every penalty function implementation has to provide

1. the evaluation of $\rho(s)$ as well as
2. its first two derivatives $\rho'(s)$, $\rho''(s)$.

All PLS solvers of section 2.2 are completely generic in the penalty function, facilitating the inclusion of new penalty functions $\rho(s)$ using the interface `penFunctions.m`:

```
% A penalty function pen(s) is a function IR^q -> IR.
%
%   [p,dp,d2p] = pen(s,psi) where psi contains additional parameters
%
% The return arguments have the following meaning:
%   p   = pen(s),
%   dp  = d   pen(s) / ds,
%   d2p = d^2 pen(s) / ds^2,
% and the following dimensions:
%   p     function value   [1x1] or vector of individual function values [qx1],
%   dp    gradient vector  [qx1],
%   d2p   Hessian diagonal [qx1] or full matrix [qxq].
%
% Currently, we have implemented in pen/pen<NAME>.m
%   Absolute value:          penAbs(s)               = abs(s),     => potLaplace
%   Smoothed absolute value: penAbsSmooth(s,ep)      = sqrt(s^2+ep),
%   Linear pen. on negative: penNegLin(s)            = max(-s,0),
%   Logarithmic:             penLogSmooth(s,nu)      = log(s^2+nu),=> potT
%   Power:                   penPow(s,d)             = abs(s)^d,   => potExpPow
%   Smoothed power:          penPowSmooth(s,d,ep)    = sqrt(s^2+ep)^d,
%   Quadratic:               penQuad(s)              = s^2/2,      => potGauss
%   Quadr. pen on negative:  penNegQuad(s)           = min(s,0)^2/2,
%   Zero:                    penZero(s)              = 0, and
%
%   Derived from a potential for use with Variational bounding (VB) inference:
%     penVB(s,pot,tau,z) = tau*b*(r-s) -log(pot(tau*r)), r=sign(s)*sqrt(s^2+z)
%   Norm penalty derived from symmetric potential to be used with VB inference:
%     penVBNorm(s,pot,tau,z,G) = -log( pot(tau*r) ), r=sqrt( G*(s^2+z) )
%
% Examples:
% >>  s=0.3; d=1.5; penPow(s,d)
% >>  s=-4;  pot='potLaplace'; tau=2; z=1.3; penVB(s,pot,tau,z)
%
%   See also POTFUNCTIONS.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2010 November 09
```

2.3.2 Implementations

The interface is currently implemented by several simple and two composed penalty function $\rho(s) = \rho(s;\boldsymbol{\psi})$ in `pen/pen<NAME>.m`. Note that the penalties can depend on additional parameters ψ.

<NAME>	Meaning	Expression	Parameters $\boldsymbol{\psi} =$
Abs	Absolute value penalty	$\rho(s;\boldsymbol{\psi}) = \lvert s \rvert$	$\emptyset$
AbsSmooth	Smoothed absolute value penalty	$\rho(s;\boldsymbol{\psi}) = \sqrt{s^2+\varepsilon}$, $\varepsilon > 0$	ε
NegLin	Linear penalty on negative part	$\rho(s;\boldsymbol{\psi}) = \max(-s, 0)$	$\emptyset$
Pow	Power penalty	$\rho(s;\boldsymbol{\psi}) = \lvert s \rvert^{\alpha}$, $\alpha > 0$	α
PowSmooth	Smoothed power penalty	$\rho(s;\boldsymbol{\psi}) = \left(\sqrt{s^2+\varepsilon}\right)^{\alpha}$, $\alpha > 0$, $\varepsilon > 0$	α, ε
Quad	Quadratic penalty	$\rho(s;\boldsymbol{\psi}) = \frac{1}{2}s^2$	$\emptyset$
NegQuad	Quadratic penalty on negative part	$\rho(s;\boldsymbol{\psi}) = \frac{1}{2}s_-^2$ where $s_- = \min(s, 0)$	$\emptyset$
LogSmooth	Smoothed logarithmic penalty	$\rho(s;\boldsymbol{\psi}) = \ln(s^2+\varepsilon)$, $\varepsilon > 0$	ε
Zero	No penalty at all	$\rho(s;\boldsymbol{\psi}) = 0$	$\emptyset$
VB	Penalty derived from potential for VB	$\rho(s;\boldsymbol{\psi}) = \beta\tau\,(\varsigma - s) - \ln \mathcal{T}\,(\tau\varsigma;\boldsymbol{\theta})$, $\varsigma = \operatorname{sign}(s)\cdot\sqrt{s^2+z}$	$(\mathcal{T},\boldsymbol{\theta})$, τ, z
VBNorm	Norm penalty from symmetric potential for VB	$\rho(\mathbf{s};\boldsymbol{\psi}) = -\ln \mathcal{T}\left(\boldsymbol{\tau} \odot \sqrt{\mathbf{G}(\mathbf{s}^2+\mathbf{z})};\boldsymbol{\theta}\right)$	$(\mathcal{T},\boldsymbol{\theta})$, $\boldsymbol{\tau}$, $\mathbf{G}$, $\mathbf{z}$

Besides simple penalty functions, we offer the penalty functions `pen/penVB*.m` transforming a potential function $\mathcal{T}\,(s;\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ into a penalty function $\rho(s)$. The penalty function `pen/penVB.m` with $z = 0$ allows to do MAP estimation $\rho(s) = -\ln \mathcal{T}\,(s;\boldsymbol{\theta})$; with $z > 0$ it allows to cast the inner loop optimisation as a PLS problem with $\rho(s) = h^*(s)$, see section 1.2. In a similar spirit, the penalty function `pen/penVBNorm.m` transforms a potential function $\mathcal{T}\,(s;\boldsymbol{\theta})$ into a penalty function on norms as needed for group potentials as detailed in section 1.2.2.

There are equivalences or redundancies among the penalty functions as listed in the following table.

Penalty function	Equivalent penalty function
`penAbs(s)`	`penPow(s,al=1)` `penVB(s,'potLaplace',tau=1,z=0)`
`penAbsSmooth(s,ep)`	`penVB(s,'potLaplace',tau=1,z=ep)`
`penLogSmooth(s,ep)`	`penVB(s,'potT',tau=1,z=0,nu=ep)`
`penPow(s,al)`	`penVB(s,'potExpPow',tau=1,z=0,al)`
`penPowSmooth(s,al,ep)`	`penVB(s,'potExpPow',tau=1,z=ep,al)`
`penQuad(s)`	`penPow(s,al=2)` `penVB(s,'potGauss',tau=1,z=0)`

The main reason for the redundancy is computation time and ease of usage. In general, the left expression is much faster to evaluate and also shorter in the code.

2.4 Potential functions $\mathcal{T}(s)$ in `pot/`

Potential functions are used to shape the Bayesian posterior of our model class (equation 1). In general, potentials have to be nonnegative $\mathcal{T}(s) \geq 0$. Depending on the approximate inference method, additional properties are needed. In order to run the respective inference algorithm, different aspects of the potential $\mathcal{T}(s)$ are required.

2.4.1 Variational bounding

For VB, section 1.2, three conditions have to be met by a potential function $\mathcal{T}(s)$:

1. Strict positivity: $\mathcal{T}(s) > 0$.
2. Symmetry: $\forall s \in \mathbb{R}\ \exists \beta \in \mathbb{R}\ f_\beta(s) = f_\beta(-s)$ where $f_\beta(s) = \mathcal{T}(s)e^{-\beta s}$. For $\mathcal{T}(s) = \mathcal{T}(-s)$, $\beta = 0$.
3. Super-Gaussianity: $g_\beta(x)$ is convex and decreasing where $g_\beta(x) = \ln \mathcal{T}(\sqrt{x}) - \beta\sqrt{x}$.

We do not require log-concavity of $\mathcal{T}(s)$ but log-concavity or equivalently convexity of $\rho(s) = -\ln \mathcal{T}(s)$ leads to a convex optimisation problem for both inference and estimation.

2.4.2 Expectation propagation

EP, section 1.4, requires Gaussian integrals $\hat{Z} = \int \mathcal{N}(s|\mu_\neg, \rho_\neg)\mathcal{T}(s)^\eta ds$ as well as derivatives w.r.t. to the cavity mean $\frac{\partial \ln \hat{Z}}{\partial \mu_\neg}$ and $\frac{\partial^2 \ln \hat{Z}}{\partial \mu_\neg^2}$. The potentials $\mathcal{T}(s)$ do not need to be symmetric, strictly positive or super-Gaussian to be compatible with the EP algorithm.

2.4.3 Interface

Every potential implementation in `pot/` has to provide for

- VB the evaluation of $\ln \mathcal{T}(s)$ as well as its first two derivatives $[\ln \mathcal{T}]'(s)$, $[\ln \mathcal{T}]''(s)$ as well as the symmetry parameter β and for
- EP the evaluation of $\ln \hat{Z}(\mu_\neg)$, $[\ln \hat{Z}]'(\mu_\neg)$ and $[\ln \hat{Z}]''(\mu_\neg)$.

All methods for inference and estimation are completely generic in the potentials, which makes it very simple to include new potentials into the toolbox as long as the following interface `potFunctions.m` is respected:

```
% A potential pot(s) is a scalar positive function IR -> IR_+.
% Applied to a vector valued input s, a potential function is understood as
% pointwise evaluation of every component s(i).
%
% We currently support two classes of inference algorithms beyond Laplace's
% approximation aka MAP estimation: Variational bounding and expectation
% propagation. They are specified by the type parameter in the general call
%   P = pot(s,theta,type,z)
% where theta contains additional (hyper)parameter(s). The input s is only
% used as s(:), that is, matrix structure is ignored.
%
% 1) Variational bounding (VB)  [for ExpPow, Gauss, Laplace, Logistic, Sech2, T]
%   P = pot(s,theta) = pot(s,theta,'VB')
%   - pot(s) Needs to be strongly super-Gaussian.
%   - pot(s) Has to be symmetric or symmetrisable i.e. there is a constant b
%            such that f(s) = pot(s)*exp(-b*s) is even or symmetric f(s)=f(-s).
%   The return argument is a matrix P of size [qx4] whose columns
%   [lp,dlp,d2lp,b] are as long as there are elements in the input s.
%     lp(s)   = log( pot(s) ),                    [simple log evaluation]
%     dlp(s)  = d   lp(s) / ds,                   [first  derivative of the log]
%     d2lp(s) = d^2 lp(s) / ds^2, and             [second derivative of the log]
%     b s.t. f(s)=f(-s), f(s) = pot(s)*exp(-b*s). [linear symmetry parameter]
%
% 2) Expectation propagation (EP)          [for Gauss, Laplace, Logistic, Sech2]
%   P = pot(s,theta,'EP',z)
%   - pot(s) Does not need to be strongly super-Gaussian or symmetric; we need
```

```
%            Gaussian expectations w.r.t. pot(s) in order to run EP.
%   The return argument is a matrix P of size [qx3] whose columns
%   [lZ,dlZ,d2lZ] are as long as there are elements in the input s.
%     lZ(s,z)   = log( \int N(t|s,z) pot(t) dt ),       [log partition function]
%     dlZ(s,z)  = d   lZ(s,z) / ds, and           [first  derivative of the log]
%     d2lZ(s,z) = d^2 lZ(s,z) / ds^2.             [second derivative of the log]
%
% We use a single matrix-valued output argument to allow very compact
% specifications of mixed potentials, see the last line of the examples below.
% This ease of specification, however, comes at the expense of some computional
% overhead since we always evaluate the full matrix P even if only the first
% column is needed.
%
% Currently, the following potentials are implemented in pot/pot<NAME>.m:
%   <NAME>         Analytic Expression     Param  Description
%   ------------------------------------------------------------------------
%   ExpPow(s,al) = exp(-|s|^al)            al>0   Exponential power distribution
%   Gauss(s)     = exp(-s^2/2)                    Gaussian          distribution
%   Laplace(s)   = exp(-abs(s))                   Laplace           distribution
%   Logistic(s)  = 1/(1+exp(-s))                  Logistic          function
%   Sech2(s)     = 1 / cosh(s)^2                  Sech-squared      distribution
%   T(s,nu)      = (1+s^2/nu)^(-nu/2-1/2)  nu>0   Student's t       distribution
%
% Examples:
% >> s=0.3;   pot = @(s) potLogistic(s,'EP');
% >> s=2.7;   nu=3; pot = @(s) potT(s,'VB',nu);
% >> s=[1;2]; al=4; pot = @(s) [potGauss(s(1),'VB'); potExpPow(s(2),'VB',al)];
%
% Another way of concatenating potentials is provided by pot/potCat.m, which
% allows concatenation by:
% >> pot = @(s,varargin) potCat(s,varargin{:},{@potT,@potGauss},{1:5,6:9});
%
%   See also PENFUNCTIONS.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2010 November 10
```

2.4.4 Implementations

The interface is currently implemented by six widely used potentials $\mathcal{T}(s) = \mathcal{T}(s;\boldsymbol{\theta})$ – all located in `pot/pot<NAME>.m`.

`<NAME>`	Meaning	Expression $\mathcal{T}(s;\boldsymbol{\theta}) =$	Parameters $\boldsymbol{\theta} =$
`ExpPow`	Exponential Power	$\exp(-\lvert s\rvert^{\alpha})$, $\alpha > 0$	α
`Gauss`	Gaussian	$\exp(-\frac{1}{2}s^2)$	$\emptyset$
`Laplace`	Laplacian	$\exp(-\lvert s\rvert)$	$\emptyset$
`Logistic`	Logistic function	$\frac{1}{1+\exp(-s)}$	$\emptyset$
`Sech2`	Sech-squared	$\frac{1}{\cosh^2(s)}$	$\emptyset$
`T`	Student's t	$\left(1+\frac{1}{\nu}s^2\right)^{-(\nu+1)/2}$	ν
`Cat`	Concatenation	$[\mathcal{T}_1(\mathbf{s}_{\mathbf{i}_1}),..,\mathcal{T}_r(\mathbf{s}_{\mathbf{i}_r})]$	$\mathcal{T}_i, \mathbf{i}_i$

Note that the potentials $\mathcal{T}(s)$ can depend on additional parameters $\boldsymbol{\theta}$. Further note, that the implementations of the potentials do not have a scale parameter τ. A scale τ can be introduced by replacing $\{\ln\mathcal{T}(s),\ [\ln\mathcal{T}]'(s),\ [\ln\mathcal{T}]''(s),\ \beta\}$ with $\{\ln\mathcal{T}(\tau s),\ \tau\cdot[\ln\mathcal{T}]'(\tau s),\ \tau^2\cdot[\ln\mathcal{T}]''(\tau s),\ \tau\cdot\beta\}$.

Attention: If a scale τ is introduced, the scale τ only occurs inside $\mathcal{T}(\cdot)$ not outside as e.g. needed for the Gaussian potential $\mathcal{N}(s|0,1) = \sigma\cdot\mathcal{N}(s|0,\sigma^2)$ or others. The reason for that is the fact that the classification potential has to assume values between 0 and 1 and hence cannot be scaled using the general Jacobi correction rule. This influences the marginal likelihood computation.

2.5 Double loop inference `dli.m` engine in `inf/`

Although all derivations are done using the variational parameter γ, we use $\boldsymbol{\pi} = \boldsymbol{\gamma}^{-1}$ in the implementation to avoid unnecessary inversions. The inference engine `inf/dli.m` uses PLS solvers in the inner

loop, see section 2.2. Outer loop updates are either approximated by `inf/diaginv_lanczos.m`, exactly computed by `inf/diaginv_woodbury.m`, approximated by `inf/diaginv_factorial.m` or estimated by sampling Papandreou and Yuille [2011] `inf/diaginv_sample.m`. The interface to the double loop algorithm is detailed in `infEngine.m`.

```
%   [m,ga,b,z,zu,nlZ] = DLI(X,y,s2,B,pot,tau,opts,G)
%
% DLI - Double Loop Inference - Bayesian Inference using a Double Loop algorithm
%       The variational criterion whose stationary point is found has the form
%         phi(ga,b) = ln|A| + h(ga,b) + \min_u R(u,ga,b), where
%         A         = X'*X/s2 + B'*diag(1./ga)*B, s=B*u,
%         h(ga,b)   = \sum_{j=1}^q h_j(ga_j,b_j)
%         R(u,ga,b) = (1/s2) * ||X*u-y||^2 + s'*diag(1./ga)*s - 2*b'*s.
%
% The probabilistic model consists of
%   (i)  Gaussian potentials y = X*u + e, e~N(0,s2*I)   and
%   (ii) non-Gaussian potentials t_i(s_i), pot(s) = \prod_i t_i(tau_i*s_i)
% leading to a posterior P(u) of the form
%    P(u) = (1/Z) N(u|Xu,s2*I) * pot(s) = N(u|m,V) where s = B*u and the
%    partition function is Z = \int  N(u|Xu,s2*I) * pot(s) du.
%    Furthermore, m is the posterior mean and V = inv(A) is the posterior
%    covariance matrix with diagonal zu = diag(V). The normaliser is represented
%    in the log domain and hence nlZ = -log(Z).
%
% OUTER LOOP: z \approx diag(B'*inv(A)*B) = var(s)
%
% opts.innerType == 'VB'
% INNER LOOP: min_u  (1/s2)*||X*u-y||^2 + 2*pen(s) where r = sign(s)*sqrt(s^2+z)
%              pen(s) = tau*b*(r-s) - ln pot(tau*r), see pen/penVB.m.
%   Our variational inference relaxation uses Gaussian individual lower bounds
%   pot(s) \ge exp( b*s -s^2/ga +h(ga)/2 ) to obtain a joint a lower bound
%     Z \ge exp(h(ga)/2) * \int  N(u|Xu,s2*I) * exp( b*s -s^2/ga ) du = Z_{VB}
%   having the form of a Gaussian integral that can be written as
%     Z_{VB} \ge C*exp( -phi(ga)/2 ), C = (2*pi)^(n/2) * (2*pi*s2)^(-m/2).
%   Here, the variational criterion phi(ga) is to be minimised w.r.t. the
%   variational parameters ga. For log-concave models, this constitutes a convex
%   variational optimisation problem (1).
%   Our approach to solve the problem uses Fenchel duality to decouple ln|A| by
%   upper bounding ln|A| \min_z z'*(1./ga) g*(z); a function that is just a sum
%   over individual components of ga. The algorithm consists of an outer loop
%   where the decoupling bound with coefficients z is refit and an inner loop
%   where the variational criterion is jointly minimised in both u and ga
%   (wich reduces to a PLS problem, pls/pls*.m).
%
% opts.innerType == 'EP'
% INNER LOOP:
%   We interleave parallel EP updating steps and posterior mean recomputations
%   to find a stationary point of the inner loop criterion.
%
% For factorial meanfield inference, use opts.innerType = 'VB' along with
% opts.outerMethod = 'factorial'.
%
% INPUT
% =====
%  X   [mxn]  measurement matrix or operator
%  y   [mx1]  measurement vector
%  s2  [1x1]  measurement variance
%  B   [qxn]  matrix or operator
%  pot        potential function handle or function name string from pot/pot*.m
%  tau [qx1]  scale parameters of the potentials
%  opts.      optimisation parameters
%       outerZinit   initial value for upper bound             [default 0.05]
%       outerGainit  initial value for variational parameter   [default 1]
%       outerNiter   number of outer loop iterations           [default 10]
%       outerMVM     number of MVMs/Lanczos vectors/..         [default 50]
```

```
%       outerVarOpts extra params for z computation            [default [] ]
%       outerMethod  z computation algorithm                 [default 'lanczos']
%                    z = diag( B*inv(A)*B' ) with A = X'*R*X + B'*P*B
%                    algorithm <AAA> located in inf/diaginv_<AAA>.m
%                    'full'      exact dense matrix based computation
%                    'lanczos'   Lanczos approximation based on MVMs
%                                varOpts.MVM = 50 per default
%                    'sample'    Monte Carlo estimate based on MVMs
%                                varOpts.NSamples = 10 and
%                                varOpts.Ncg = 20 per default
%                    'woodbury'  exact dense matrix computation for B=I and P, R
%                                cheaply invertable and X numeric
%                    'factorial' approximate computation for mean field
%                                inference
%       outerOutput  flag saying whether some output is shown  [default false]
%       innerMVM     number of inner loop MVMs or CG steps     [default 50]
%       innerIt      number of inner Newton steps for plsTN    [default 15]
%       innerOutput  flag saying whether some output is shown  [default false]
%       innerType    which inference method EP or VB           [default 'VB']
%       innerVBpls   inner loop PLS solver                   [default 'plsLBFGS']
%       => the opts struct is passed to the PLS solver which
%          allows to pass additional arguments to it
%       innerEPeta   scalar parameter for fractional EP        [default 1]
%                    only possible for potGauss and potLaplace
%  G    [qtxq] grouping matrix (for VB only)                    [default eye(q)]
%               this changes the inner loop penaliser to
%               pen(s) = -log( pot(tau*r) ) where r = sqrt( G*(s^2 + z) ).
%
% OUTPUT
% ======
%  m      [nx1]  posterior mean estimate                m = mean(u)
%  ga     [qx1]  optimal value of the variational width parameters
%  b      [qx1]  optimal value of the variational position parameters
%  z      [qx1]  posterior marginal variance estimate  z  = var(s) = var(B*u)
%  zu     [nx1]  posterior marginal variance estimate  zu = var(u)
%  nlZ    [1x1]  approxmation to the negative log marginal likelihood -log(Z)
%  Q      [n,k]  orthogonal (dense) matrix of Lanczos vectors, Q'*Q = I
%  T      [k,k]  diagonal (dense) matrix, T = Q'*A*Q
%  Note that Q and T are empty for outerMethod='woodbury' and n>m as well as for
%  outerMethod='sample'. For outerMethod='woodbury' and n<m as well as for
%  outerMethod='full', we have n=k and hence V = inv(A) = Q*inv(T)*Q'. For
%  outerMethod='lanczos', this relation holds only approximately. In case of
%  outerMethod='factorial', the relation is exact but A is a diagonal matrix.
%
%   See also PENFUNCTIONS.M, POTFUNCTIONS.M, INF/DLI.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2012 January 13
```

2.5.1 Auxiliary routines for the outer loop

The outer loop update (see section 1.2.1) (approximately) computes the marginal variance i.e. the diagonal $\mathbf{z} = \mathrm{dg}(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top)$ of the inverse $\mathbf{A} = \sigma^{-2}\mathbf{X}^\top\mathbf{X} + \mathbf{B}^\top\mathbf{\Gamma}^{-1}\mathbf{B}$ of the covariance matrix $\mathbf{V} = \mathbf{A}^{-1}$. Besides exact dense matrix computations in `inf/diaginv_full.m`, the toolbox offers three complementary approximation functions to accomplish this task. One exploits successive matrix vector multiplications (MVM) to approximate $\mathbf{z}$ using the Lanczos algorithm `inf/diaginv_lanzcos.m`, another one uses MVMs to drive a Monte Carlo sampler [Papandreou and Yuille, 2011] `inf/diaginv_sample.m` and the third one `inf/diaginv_woodbury.m` concentrates on the special case $\mathbf{B} = \mathbf{I}$. In that case, where $\mathbf{z} = \mathrm{diag}(\mathbf{A}^{-1})$, we can – depending on the sizes m and n of $\mathbf{X}$ – exploit the Woodbury formula

$$\mathbf{A}^{-1} = \left(\mathbf{X}^\top\mathbf{X}/\sigma^2 + \mathbf{\Gamma}^{-1}\right)^{-1} = \mathbf{\Gamma} - \mathbf{\Gamma}\mathbf{X}^\top(\sigma^2\mathbf{I} + \mathbf{X}\mathbf{\Gamma}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{\Gamma}$$

to compute $\mathbf{z}$ exactly provided that either m or n is small.

The other implementations `inf/diaginv_{factorial,full,lanzcos,sample}.m` consider a more general matrix $\mathbf{A} = \mathbf{X}^\top\mathbf{R}\mathbf{X} + \mathbf{B}^\top\mathbf{P}\mathbf{B}$ where $\mathbf{R}$ and $\mathbf{P}$ can either be specified by a

- positive definite square matrices: $\mathbf{R} \in \mathbb{R}^{m \times m}$, $\mathbf{P} \in \mathbb{R}^{q \times q}$
- positive vectors representing the diagonals: $\mathbf{R} = \text{diag}(\mathbf{r})$, $\mathbf{P} = \text{diag}(\mathbf{p})$, $\mathbf{r} \in \mathbb{R}_+^m$, $\mathbf{p} \in \mathbb{R}_+^q$, or
- positive numbers to scaling the identity matrix: $\mathbf{R} = r\mathbf{I}$, $\mathbf{P} = p\mathbf{I}$, $r \in \mathbb{R}_+$, $p \in \mathbb{R}_+^q$.

In the outer loop, we use $\mathbf{R} = \sigma^{-2}\mathbf{I}$ and $\mathbf{P} = \mathbf{\Gamma}^{-1}$.

3 Installation and compilation of MEX code

Before using the glm-ie toolbox, you should run startup.m which adds some entries to the path variable.

In order to use the L-BFGS minimiser to solve the PLS optimisation problem, you have to compile Peter Carbonetto's Matlab interface for L-BFGS-B. The challenge here is the Fortran 77 code. We provide a Makefile suitable for Linux 32/64 bit and Mac whenever you have g77 properly installed. Compilation is done by first adapting pls/lbfgsb/Makefile to your computing environment. In any case, you need to provide $MATLAB_HOME which can be found by the commands locate matlab, find / -name "matlab" or the like. You can choose between two compilation modes:

1. using the mex utility by Matlab which is the default mode
   (a) provide the variable $MEX, then type
   (b) >> cd pls/lbfgsb
   (c) >> make mex
   (d) >> cd ../..
2. without the mex utility by Matlab
   (a) provide the variables $MEX_SUFFIX and $MATLAB_LIB, then type
   - >> cd pls/lbfgsb
   - >> make nomex
   - >> cd ../..

Notes on Ubuntu

In Ubuntu 10.04 LTS, the libg2c library needed for both compilation modes is not included per default anymore. You can check this by ls /usr/lib/libg2c.* which produces an empty output in this case on your machine. You then want to install the packages gcc-3.4-base[4] and libg2c0[5]. After installation, you have to create a symbolic link by cd /usr/lib and ln -s libg2c.so.0 libg2c.so. In combination with the fort77 package, you should be able to compile the code.

Notes on Windows

Compilation under Windows is rather tricky but there is a way of doing it[6] as shown by Guillaume Jacquenot. A list of compilers can be found at http://www.mathworks.com/support/compilers/R2010a/. Steps involved:

- Install MinGW[7] (including gfortran compiler) at C:\mingw.
- If the target was Matlab 64-bit, we would probably need MinGW-w64 instead?
- Download gnumex[8], unzip at C:\gnumex, and create Fortran/C mexopts.bat files for use with MinGW.
- Add C:\mingw\bin and C:\gnumex in the PATH and reboot.
- Create and run the pls/make_lbfgsb_gnumex_windows.m script; replace the filenames for the mexopts.bat files accordingly.
- Use MS Dependency walker[9] to debug missing DLL dependencies.

[4] http://packages.ubuntu.com/jaunty/gcc-3.4-base
[5] http://packages.ubuntu.com/jaunty/libg2c0
[6] http://www.cs.ubc.ca/~pcarbo/Compile_LBFGSB_on_Windows.txt
[7] http://www.mingw.org/
[8] http://gnumex.sourceforge.net/
[9] http://www.dependencywalker.com

References

Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8:141–148, 1988. 10

Patrick Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. Springer, 2010. URL `http://arxiv.org/abs/0912.3522`. 10

Tom Goldstein and Stanley Osher. The split bregman method for l1 regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009. URL `ftp://ftp.math.ucla.edu/pub/camreport/cam08-29.pdf`. 10

James Miskin and David J.C. MacKay. Ensemble learning for blind image separation and deconvolution. In *Advances in Independent Component Analysis*, 2000. 4

Hannes Nickisch and Matthias Seeger. Convex variational Bayesian inference for large scale generalized linear models. In *Proceedings of the 26th International Conference on Machine Learning*, 2009. 2

George Papandreou and Alan Yuille. Efficient variational inference in large-scale Bayesian compressed sensing. In *Proc. IEEE Workshop on Information Theory in Computer Vision and Pattern Recognition (in conjunction with ICCV)*, 2011. 15, 16

Matthias W. Seeger and Hannes Nickisch. Large scale variational inference and experimental design for sparse generalized linear models. Technical Report 175, Max Planck Institute for Biological Cybernetics, 9 2008. 2

Matthias W. Seeger and Hannes Nickisch. Large scale variational inference and experimental design for sparse generalized linear models. *SIAM Journal on Imaging Sciences*, 4(1):166–199, 2011. URL `http://dx.doi.org/10.1137/090758775`. 2

Matthias W. Seeger, Hannes Nickisch, Rolf Pohmann, and Bernhard Schölkopf. Bayesian experimental design of magnetic resonance imaging sequences. In *Advances in Neural Information Processing Systems 21*, pages 1441–1448, 2009. 2

Matthias W. Seeger, Hannes Nickisch, Rolf Pohmann, and Bernhard Schölkopf. Optimization of k-space trajectories for compressed sensing by bayesian experimental design. *Magnetic Resonance in Medicine*, 63(1):116–126, January 2010. 2

`fnlCg.m`. by Michael Lustig, August 2007. URL `http://www.stanford.edu/~mlustig/SparseMRI.html`. 10

`fwtn`. by Hannes Nickisch, March 2010. URL `http://mloss.org/software/view/242/`. 7

L-BFGS-B. by Ciyou Zhu, Richard Byrd and Jorge Nocedal., September 1997. URL `http://www.eecs.northwestern.edu/~nocedal/lbfgsb.html`. 10

Matlab interface for L-BFGS-B. by Peter Carbonetto, May 2007. URL `http://www.cs.ubc.ca/~pcarbo/lbfgsb-for-matlab.html`. 18

`minimize.m`. by Carl E. Rasmussen, September 2006. URL `http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/`. 10

`mrics.m`. by Tom Goldstein, December 2008. URL `http://mloss.org/software/view/242/`. 10

`nufft`. by Jeff Fessler, 2001. URL `http://www.eecs.umich.edu/~fessler/irt/irt/nufft/`. 7

Marcel van Gerven, Botond Cseke, Floris de Lange, and Tom Heskes. Efficient Bayesian multivariate fMRI analysis using a sparsifying spatio-temporal prior. *Neuroimage*, 50:150–161, 2010. 5

David Wipf and Srikantan Nagarajan. A new view of automatic relevance determination. In *Advances in Neural Information Processing Systems 20*, 2008. 2