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

Hannes Nickisch, MPI for Biological Cybernetics, Tübingen, Germany

August 10, 2010

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 3.2.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

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

Both regression and classification models fall into the scope of the framework. Nickisch and Seeger [2009], Seeger, Nickisch, Pohmann, and Schölkopf [2009], Seeger and Nickisch [2008]

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

$$\arg\min_{\mathbf{u}} \frac{1}{2} \|\mathbf{X}\mathbf{u} - \mathbf{y}\|^2 + \lambda \cdot \rho(\mathbf{s}),\ \mathbf{s} = \mathbf{B}\mathbf{u},\ \lambda \in \mathbb{R}_+ \tag{2}$$

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

1.2 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

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

where

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

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 $\mathbf{V} = \mathbf{A}^{-1}$ 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 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

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

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

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

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

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

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

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

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

1.3 Double loop algorithm

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

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

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 $\rho_{\mathrm{VB}}(\mathbf{s}) = h^*(\mathbf{s})$ and weight $\lambda = \sigma^2$.

1.4 Package organisation

Besided the illustrating examples (section **??**) 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
`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 objects in `mat/`

For the `glm-ie` toolbox, a matrix $\mathbf{A}$ is completely specified by its MVM (matrix vector multiplication) $\mathbf{Ax}$ with a vector $\mathbf{x}$. We make use of Matlab's object oriented programming facilities in order to use the same code for both dense matrices or matrices only implicitely specified through their MVM.

Note on Octave compatibility

There is an issue with the way Octave 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 not create full copies of `A` – leading to essentially no memory overhead.

2.1.1 Interface

Matlab requires several functions for a matrix object $\mathbf{A}$ (named `A` in the code) to be present in a directory `@A/` that are called whenever the interpreter encounters expressions like `A*x` for $\mathbf{Ax}$, `A'*x` (and equivalently `A.'*x`) for $\mathbf{A}^\top\mathbf{x}$, `size(A,2)` or the like.

1. `A.m` is the constructor for the matrix object and
2. `mtimes.m` implements the MVM depending on the transpose flag `A.ctransp`.

We have seven generic functions shared by all matrix objects in the `glm-ie` toolbox:

1. `ctranspose.m` flips the flag `A.ctransp` that tells `mtimes.m` whether to compute $\mathbf{Ax}$ or $\mathbf{A}^\top\mathbf{x}$,
2. `transpose.m` is equivalent to `ctranspose.m`,
3. `isempty.m` indicates whether there is no entry in $\mathbf{A}$,
4. `disp.m` displays some information about the size and the class of $\mathbf{A}$,
5. `numel.m` counts the number of elements in $\mathbf{A}$,
6. `full.m` returs the full matrix $\mathbf{A}$ and
7. `size.m` returns the size of the matrix `A` as stored in the struct field `A.sz` by the constructor `A.m`.

2.1.2 Implementations

The `glm-ie` toolbox currently implements six matrix objects $\mathbf{A}$ whose code is located in the directories `mat/@Amat/` and its subdirectories.

`A`	$\mathbf{A}$	Meaning	Formal $\mathbf{Ax} \equiv$	$\mathcal{O}(n) =$
`cnv2`	$\mathbf{K}$	Convolution with kernel $\mathbf{k}$ by Michael Hirsch	$\mathbf{k} \star \mathbf{x}$	n
`fd2`	$\mathbf{D}$	Finite derivatives in both directions	$[x_{i+1} - x_i]_i$	n
`fftline2`	$\mathbf{F_l}$	Partial Fourier, single rows	$(\mathbf{Fx})_\mathbf{l}$ subset	$n \cdot \ln n$
`fftmask2`	$\mathbf{F_M}$	Partial Fourier, single points	$(\mathbf{Fx})_\mathbf{M}$ subset	$n \cdot \ln n$
`wav`	$\mathbf{W}$	Fast wavelet transform `fwtn`	$\mathbf{Wx}$	n
`cat`	$\mathbf{B}$	Concatenation of two matrices $\mathbf{L}$, $\mathbf{R}$	$\begin{bmatrix}\mathbf{Lx}\\ \mathbf{Rx}\end{bmatrix}$	1

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 1.3) of our double loop variational inference method.

2.2.1 Interface

We provide a generic interface in `plsAlgorithms.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) algorithm is a program solving the
% minimisation problem
%   phi(u) = 1/(2*lambda) * ||X*u-y||_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  [n,1]  initial vector
%    X   [m,n]  matrix or operator
%    y   [m,1]  vector
%    B   [q,n]  matrix or operator
%    opt.       optimisation parameters
%      nMVM     maximal number of steps = matrix vector multiplications ...
%                            with A=X'*X+B'*D*B, D is 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   [n,1]  optimal solution
%    phi [1,1]  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
%
% 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,'penFromPot','potLaplace',tau,z)
%
%   See also PENFUNCTIONS.M, POTFUNCTIONS.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2010 July 27
```

2.2.2 Implementations

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

<NAME>	Meaning
CG	Conjugate gradients (CG) using Carl E. Rasmussen's code `minimize.m`.
CGBT	CG with backtracking line search using Armijo's rule `fnlCg.m`.
LBFGS	Limited memory Broyden–Fletcher–Goldfarb–Shanno quasi Newton.
TN	Truncated Newton with CG approximated Newton direction.

2.2.3 Auxiliary routines

- In `pls/plsCG.m`, the conjugate gradient optimiser `minimize.m` 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`.
- Finally, we have a truncated Newton procedure `pls/plsTN.m` running. 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). 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 scalar function IR -> IR.
%
%   [p,dp,d2p] = pen(s,psi) where psi contains additional parameters
%
% The return arguments are of the same size as s and have the following meaning:
%   p   =     pen(s),
%   dp  = d   pen(s) / ds, and
%   d2p = d^2 pen(s) / ds^2.
%
% Currently, we have implemented in pen/pen<NAME>.m
%   Logarithmic:            penLog(s,nu)            = log(s^2+nu),
%   Power:                  penPow(s,d)             = abs(s)^d,
%   Quadratic:              penQuad(s)              = s^2/2,
%   Zero:                   penZero(s)              = 0, and
%   Derived from potential: penFromPot(s,pot,tau,z) = ...
%                           tau*b*(r-s) -log( pot(tau*r) ), r=sign(s)*sqrt(s^2+z)
%
% Examples:
% >>  s=0.3; d=1.5; penPow(s,d)
% >>  s=-4;  pot='potLaplace'; tau=2; z=1.3; penFromPot(s,pot,tau,z)
%
%   See also POTFUNCTIONS.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2010 August 09
```

2.3.2 Implementations

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

`<NAME>`	Meaning	Expression $\rho(s;\boldsymbol{\psi}) =$	Parameters $\boldsymbol{\psi} =$
`Log`	Logarithmic penalty	$\ln(s^2+\nu)$, $\nu > 0$	ν
`Pow`	Power penalty	$\lvert s\rvert^d$, $d > 0$	d
`Quad`	Quadratic penalty	s^2	$\emptyset$
`Zero`	No penalty at all	0	$\emptyset$
`FromPot`	Penalty from potential	$\beta\tau(\varsigma - s) - \ln \mathcal{T}(\tau\varsigma;\boldsymbol{\theta})$, $\varsigma = \mathrm{sign}(s)\cdot\sqrt{s^2+z}$	$(\mathcal{T},\boldsymbol{\theta})$, τ, z

Besides simple penalty functions, we offer the penalty function `pen/penFromPot.m` transforming a potential function $\mathcal{T}(s;\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ into a penalty function $\rho(s)$. This penalty function is allows to cast the inner loop optimisation as a PLS problem with $\rho(s) = h^*(s)$, see section 1.2. Used with $z = 0$ and $\tau = 1$, `pen/penFromPot.m` allows to do MAP estimation $\rho(s) = -\ln \mathcal{T}(s;\boldsymbol{\theta})$.

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

Potential functions are used to shape the Bayesian posterior of our model class (equation 1). Three conditions have to be met by a proper potential function $\mathcal{T}(s)$:

1. 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.

2.4.1 Interface

Every potential implementation in `pot/` has to provide

1. the evaluation of $\ln \mathcal{T}(s)$ as well as its first two derivatives $[\ln \mathcal{T}]'(s)$, $[\ln \mathcal{T}]''(s)$ and
2. the symmetry parameter β.

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 and strongly super Gaussian
% function IR -> IR_+ that can be symmetrised i.e. there is a number b
% such that f(s) = pot(s)*exp(-b*s) is even.
%
%   P = pot(s,theta) where theta contains additional parameters
%
% The return argument is a matrix of size [px4] where the columns do have the
% following meaning:
%   lp = P(:,1), dlp = P(:,2), d2lp = P(:,3), b = P(:,4), where
%
%   (i)   lp(s)   = log( pot(s) ),
%   (ii)  dlp(s)  = d   lp(s) / ds,
%   (iii) d2lp(s) = d^2 lp(s) / ds^2, and
%   (iv)  f(s) = pot(s)*exp(-b*s) is symmetric, i.e. f(s) = f(-s).
%
% We can split the matrix P by the command
%   [lp,dlp,d2lp,b] = col(P).
%
% Currently, we have implemented in pot/pot<NAME>.m
%   Gaussian:                 potGauss(s)       = exp(-s^2/2),
%   Sech-squared:             potSech2(s)       = 1 / cosh(s)^2,
%   Laplace:                  potLaplace(s)     = exp(-abs(s)),
%   Student's t, nu>0:        potT(s,nu)        = (1 + s^2/nu)^(-nu/2-1/2),
%   Exponential power, al>0: potExpPower(s,a)  = exp(-|s|^al), and
%   Logistic function:        potLogistic(s)    = 1/(1+exp(-s)).
%
% Examples:
% >>  s=0.3; potLogistic(s)
% >>  s=2.7; nu=3; potT(s,nu)
%
%   See also PENFUNCTIONS.M.
%
% (c) by Hannes Nickisch, MPI for Biological Cybernetics, 2010 July 27
```

2.4.2 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 $\boldsymbol{\theta} =$
Gauss	Gaussian	$\exp(-\frac{1}{2}s^2)$	$\emptyset$
Laplace	Laplacian	$\exp(-\lvert s\rvert)$	$\emptyset$
Sech2	Sech-squared	$\frac{1}{\cosh^2(s)}$	$\emptyset$
Logistic	Logistic function	$\frac{1}{1+\exp(-s)}$	$\emptyset$
ExpPower	Exponential Power	$\exp(-\lvert s\rvert^\alpha)$, $\alpha > 0$	α
T	Student's t	$\left(1+\frac{1}{\nu}s^2\right)^{-(\nu+1)/2}$	ν

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 $\{\ln\mathcal{T}(\tau s),\ \tau\cdot[\ln\mathcal{T}]'(\tau s),\ \tau^2\cdot[\ln\mathcal{T}]''(\tau s),\ \tau\cdot\beta\}$.

2.5 Double loop inference engine in `inf/`

Although all derivations are done using the variational parameter $\boldsymbol{\gamma}$, we use $\boldsymbol{\pi} = \boldsymbol{\gamma}^{-1}$ in the implementation to avoid unnecessary inversions.

vbidl.m

Innner loops are done by `pls/`, see section 2.2.

2.5.1 Auxiliary routines for the outer loop

The outer loop update (see section 1.3) (approximately) computes the marginal variance i.e. the diagonal $\mathbf{z} = \mathrm{dg}(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top)$ of the inverse $\mathbf{A} = \sigma^{-2}\mathbf{X}^\top\mathbf{X} + \mathbf{B}^\top\boldsymbol{\Gamma}^{-1}\mathbf{B}$ of the covariance matrix $\mathbf{V} = \mathbf{A}^{-1}$. The toolbox offers two complementary functions to accomplish this task. One exploits successive matrix vector multiplications (MVM) to approximate $\mathbf{z}$ using the Lanczos algorithm `inf/diaginv_lanzcos.m` and the other 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

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

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

The two implementations `inf/diaginv_{lanzcos,woodbury}.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: $\mathbf{R} = \mathrm{diag}(\mathbf{r})$, $\mathbf{P} = \mathrm{diag}(\mathbf{p})$, $\mathbf{r} \in \mathbb{R}_+^m$, $\mathbf{p} \in \mathbb{R}_+^q$, 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 $\mathbf{P} = \boldsymbol{\Gamma}^{-1}$.

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. A list of compilers can be found at http://www.mathworks.com/support/compilers/R2010a/. 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 whant to install the packages gcc-3.4-base[3] and libg2c0[4]. After installation, you have to create a symbolic link by cd /usr/lib and ln -s libg2c.so.0 libg2c.so.

References

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

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, 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

fnlCg.m. by Michael Lustig, August 2007. URL http://www.stanford.edu/~mlustig/SparseMRI.html. 6

fwtn. by Hannes Nickisch, March 2010. URL http://mloss.org/software/view/242/. 4

L-BFGS-B. by Ciyou Zhu, Richard Byrd and Jorge Nocedal., September 1997. URL http://www.eecs.northwestern.edu/~nocedal/lbfgsb.html. 6

Matlab interface for L-BFGS-B. by Peter Carbonetto, May 2007. URL http://www.cs.ubc.ca/~pcarbo/lbfgsb-for-matlab.html. 10

minimize.m. by Carl E. Rasmussen, September 2006. URL http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/. 6

[3] http://packages.ubuntu.com/jaunty/gcc-3.4-base
[4] http://packages.ubuntu.com/jaunty/libg2c0