Thermodynamic Variational Laplace in Generalised Coordinates

Description Equations for `fitVariationalLaplaceThermo_GC.m`

Dr Alexander D. Shaw
Computational Psychiatry and Neuropharmacological Systems Lab
Department of Psychology, University of Exeter, UK
`a.d.shaw@exeter.ac.uk`

November 18, 2025

1 Overview

We consider a latent-variable model with parameters $m \in \mathbb{R}^P$ and observations $y \in \mathbb{R}^T$:

$$y_t = f_t(m) + \varepsilon_t, \qquad t = 1, \ldots, T, \tag{1}$$

where $f(m) = (\hat{y}_1, \ldots, \hat{y}_T)^\top$ denotes model predictions and ε_t are residuals.

Classical Variational Laplace (VL) constructs a Gaussian approximation to the posterior

$$p(m \mid y) \approx q(m) = \mathcal{N}(m; \mu, \Sigma), \tag{2}$$

by maximising a free energy / ELBO functional with respect to the variational mean μ and covariance Σ.

The routine `fitVariationalLaplaceThermo_GC.m` extends this to *generalised coordinates* of motion. Instead of only fitting the signal y itself, we also fit its discrete derivatives $Dy, D^2y, \ldots, D^qy$, where D is a finite-difference derivative operator on the sample axis and q is the highest order. Intuitively, we ask the model to match both trajectories *and* local dynamics, which stabilises optimisation and flattens the ELBO landscape.

At a high level, the routine:

1. Constructs discrete derivative operators D^k via finite differences.
2. Stacks residuals across orders $r^{(k)} = D^k y - D^k \hat{y}(m)$.
3. Weights orders via a diagonal metric Γ implementing a Sobolev-like norm.
4. Uses a heteroscedastic, t-like noise model in this GC space.
5. Forms a Gauss–Newton curvature in GC space.
6. Combines this with a smoothness prior over parameters and low-rank + diagonal precision.

2 Generalised Coordinates: Discrete Derivative Operators

Let $y \in \mathbb{R}^T$ be the observed data (vectorised in the code via `y = y(:)`). We define a discrete derivative operator $D \in \mathbb{R}^{T \times T}$ acting along the sample axis. In the default central-difference scheme (`'fd2'`), we have

$$D = \frac{1}{\Delta t} \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & \cdots & 0 \\ 0 & -\frac{1}{2} & 0 & \ddots & 0 \\ \vdots & & & \ddots & \frac{1}{2} \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}, \tag{3}$$

with simple boundary fixes to replicate or mirror the first and last samples (see `apply_boundary` in the code).

Higher-order discrete derivatives are powers of this operator:

$$D^0 = I_T, \qquad D^1 = D, \qquad D^2 = D^2, \quad \dots, \quad D^q = D^q. \tag{4}$$

The helper

```
E = build_gc_operators(T, q, dt, gc.op, gc.boundary);
% E{1} = I, E{2} = D, ..., E{q+1} = D^q
```

returns a cell array

$$E_1 = I_T, \quad E_2 = D, \quad \dots, \quad E_{q+1} = D^q. \tag{5}$$

For any $z \in \mathbb{R}^T$, the k-th order discrete derivative is

$$z^{(k)} := E_{k+1} z = D^k z, \qquad k = 0, \dots, q. \tag{6}$$

3 Stacking Residuals in Generalised Coordinates

Let

$$\hat{y}(m) = f(m) \in \mathbb{R}^T \tag{7}$$

denote the model prediction at the current parameter estimate m. The zeroth-order residual is

$$r^{(0)} = y - \hat{y} \in \mathbb{R}^T. \tag{8}$$

For higher orders $k = 1, \dots, q$, we define

$$r^{(k)} = y^{(k)} - \hat{y}^{(k)} = E_{k+1} y - E_{k+1} \hat{y} \in \mathbb{R}^T. \tag{9}$$

We then stack all orders into a single GC residual vector

$$R = \begin{bmatrix} r^{(0)} \\ r^{(1)} \\ \vdots \\ r^{(q)} \end{bmatrix} \in \mathbb{R}^n, \qquad n = (q+1)T. \tag{10}$$

In matrix form this can be written as

$$R = S\,(y - \hat{y}), \qquad S = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{q+1} \end{bmatrix} \in \mathbb{R}^{(q+1)T \times T}. \tag{11}$$

Code mapping. In the MATLAB code:

```
r0 = y - yhat;
R  = r0;

for kOrd = 1:q
    y_k    = E{kOrd+1} * y;
    yhat_k = E{kOrd+1} * yhat;
    rk     = y_k - yhat_k;
    R      = [R; rk];
end
```

exactly implements $R = [r^{(0)}; r^{(1)}; \dots; r^{(q)}]$.

4 Sobolev Metric Over Orders: Γ

We introduce per-order weights $\lambda_k > 0$, $k = 0, \dots, q$, to construct a Sobolev-like norm over orders. By default,

$$\lambda_k = \frac{1}{(\Delta t)^{2k}}, \qquad k = 0, \dots, q, \tag{12}$$

corresponding to the code path where `opts.gc.lambda` is empty.

We collect these into a diagonal matrix

$$\Lambda = \operatorname{diag}(\lambda_0, \dots, \lambda_q) \in \mathbb{R}^{(q+1)\times(q+1)}. \tag{13}$$

To apply the same weights at each time point, we define

$$\Gamma = I_T \otimes \Lambda \in \mathbb{R}^{(q+1)T\times(q+1)T}, \tag{14}$$

and its principal square root

$$\Gamma^{1/2} = I_T \otimes \Lambda^{1/2}. \tag{15}$$

In code:

```
Gamma     = kron(spdiags(lam,0,q+1,q+1), speye(T)); % (q+1)T × (q+1)T
sqrtGamma = spdiags(sqrt(repmat(lam,T,1)),0,(q+1)*T,(q+1)*T);
```

where `lam` is the vector $(\lambda_0, \dots, \lambda_q)^\top$.

For a stacked vector $z \in \mathbb{R}^{(q+1)T}$, the Sobolev-type norm is

$$\|z\|_\Gamma^2 := z^\top \Gamma z = \sum_{k=0}^{q} \lambda_k \, \|z^{(k)}\|_2^2, \tag{16}$$

where $z^{(k)}$ is the block corresponding to order k.

Applied to the residuals R, this becomes a GC-weighted norm

$$\|R\|_\Gamma^2 = R^\top \Gamma R. \tag{17}$$

5 Heteroscedastic Noise in GC Space

We place a heteroscedastic Gaussian noise model on the GC residuals:

$$R_i \sim \mathcal{N}(0, \sigma_i^2), \qquad i = 1, \dots, n = (q+1)T. \tag{18}$$

Let

$$\Sigma = \operatorname{diag}(\sigma_1^2, \dots, \sigma_n^2) \in \mathbb{R}^{n\times n} \tag{19}$$

denote the diagonal noise covariance in GC space.

The routine uses a t-like update for σ_i^2 derived from an inverse-Gamma prior over precisions. The update has the form

$$\sigma_i^2 = \frac{R_i^2 + \beta}{\nu + \frac{1}{2}R_i^2}, \tag{20}$$

with small constants $\beta > 0$ and $\nu > 0$ controlling robustness.

In the code:

```
sigma2 = max(epsilon, (R.^2 + beta) ./ (nu + R.^2/2));
```

implements this elementwise.

We define the whitened residual

$$\tilde{R} = \Sigma^{-1/2} R, \qquad \tilde{R}_i = \frac{R_i}{\sqrt{\sigma_i^2}}. \tag{21}$$

In the code:

```
Wsqrt  = spdiags(1./sqrt(max(epsilon,sigma2)), 0, n, n); % ^{-1/2}
RW     = Wsqrt * R;                                       % ^{-1/2} R
RWg    = sqrtGamma * RW;                                  % ^{1/2} ^{-1/2} R
```

so that the GC-weighted residual is

$$\tilde{R}_g = \Gamma^{1/2} \Sigma^{-1/2} R. \tag{22}$$

The negative log-likelihood term (up to constants) is

$$-\log p(R \mid m) \approx \frac{1}{2} R^\top \Sigma^{-1} R + \frac{1}{2} \sum_{i=1}^{n} \log(2\pi\sigma_i^2) + \frac{1}{2} \sum_{k=0}^{q} \log \lambda_k. \tag{23}$$

In the code this is:

```
logL_lik = -0.5 * (sum((R.^2)./sigma2) + sum(log(2*pi*sigma2))) ...
           -0.5 * sum(log(lam + eps));
```

6 Jacobian in Generalised Coordinates

Let the Jacobian of the model in signal space be

$$J_0(m) = \frac{\partial \hat{y}}{\partial m} \in \mathbb{R}^{T \times P}, \tag{24}$$

so that for small parameter perturbations δm we have

$$\hat{y}(m + \delta m) \approx \hat{y}(m) + J_0(m)\, \delta m. \tag{25}$$

The routine estimates J_0 by central differences:

```
J0 = computeJacobian(@(mm) f(mm), m, T);
```

i.e.

$$[J_0]_{t,p} \approx \frac{\hat{y}_t(m + \varepsilon e_p) - \hat{y}_t(m - \varepsilon e_p)}{2\varepsilon}. \tag{26}$$

To lift the Jacobian into generalised coordinates, we apply the same derivative operators E_k to each column of J_0:

$$J^{(0)} = J_0, \tag{27}$$

$$J^{(k)} = E_{k+1} J_0 = D^k J_0 \in \mathbb{R}^{T \times P}, \qquad k = 1, \dots, q. \tag{28}$$

We then stack these blocks:

$$J_{\mathrm{GC}} = \begin{bmatrix} J^{(0)} \\ J^{(1)} \\ \vdots \\ J^{(q)} \end{bmatrix} = \begin{bmatrix} E_1 J_0 \\ E_2 J_0 \\ \vdots \\ E_{q+1} J_0 \end{bmatrix} \in \mathbb{R}^{(q+1)T \times P}. \tag{29}$$

Equivalently,

$$J_{\mathrm{GC}} = S J_0. \tag{30}$$

Code mapping.

```
J_gc = J0;
for kOrd = 1:q
    J_gc = [J_gc; E{kOrd+1}*J0];
end
```

implements the stacking of $J^{(k)}$.

We then whiten and GC-weight the Jacobian:

$$\tilde{J} = \Gamma^{1/2}\Sigma^{-1/2} J_{\mathrm{GC}} \in \mathbb{R}^{(q+1)T\times P}. \tag{31}$$

In the code:

```
JW  = Wsqrt * J_gc;   % ^{-1/2} J_gc
JWG = sqrtGamma * JW; % ^{1/2} ^{-1/2} J_gc
```

so that $\tilde{J} =$ `JWG`.

7 Prior and Smoothness Regularisation

We assume a Gaussian prior on m of the form

$$p(m) = \mathcal{N}(m; m_0, S_0 + K), \tag{32}$$

where S_0 is a user-supplied prior covariance (e.g. from a DCM parameterisation) and K is a smoothness covariance enforcing continuity over neighbouring parameter indices.

The helper

```
K = computeSmoothCovariance(P, 2);
```

implements a squared-exponential covariance over indices $i, j = 1, \ldots, P$:

$$K_{ij} = \exp\Big(-\frac{(i-j)^2}{2\ell^2}\Big) + \epsilon\,\delta_{ij}, \tag{33}$$

with length-scale $\ell = 2$ and small jitter ϵ.

The prior precision is

$$H_{\mathrm{prior}} = (S_0 + K)^{-1}. \tag{34}$$

In the code this is computed via a Cholesky factorisation:

```
A = (S0 + K);
R = chol(A,'lower');
H_prior = R'\(R\eye(P)); % (S0 + K)^{-1}
```

The prior contribution to the ELBO is

$$\log p(m) = -\frac{1}{2}(m - m_0)^{\top} H_{\mathrm{prior}}(m - m_0) + \mathrm{const}, \tag{35}$$

implemented as

```
logL_prior = -0.5 * ((m - m0)' * H_prior * (m - m0));
```

8 Gauss–Newton Update, Low-Rank Precision and ELBO

8.1 Gauss–Newton curvature in GC space

Under a Gaussian approximation, the negative log-likelihood in GC space is locally

$$\frac{1}{2}\tilde{R}_g^\top \tilde{R}_g \approx \frac{1}{2}(m-\mu)^\top H(m-\mu), \tag{36}$$

where the Gauss–Newton curvature is

$$H = \tilde{J}^\top \tilde{J} = J_{\mathrm{GC}}^\top \Sigma^{-1/2}\Gamma\Sigma^{-1/2}J_{\mathrm{GC}}. \tag{37}$$

In the code this is

```
H = JWG' * JWG;
```

with gradient

$$g = \tilde{J}^\top \tilde{R}_g - H_{\mathrm{prior}}(m-m_0). \tag{38}$$

Implemented as:

```
g = JWG' * RWg - H_prior*(m - m0);
```

The full posterior precision is

$$H_{\mathrm{post}} = H + H_{\mathrm{prior}}. \tag{39}$$

8.2 Low-rank + diagonal representation

For scalability, the routine approximates H_{post} by a low-rank plus diagonal structure:

$$H_{\mathrm{post}} \approx V_r V_r^\top + \mathrm{diag}(D_r), \tag{40}$$

where $V_r \in \mathbb{R}^{P\times k}$ captures a leading eigenspace and $\mathrm{diag}(D_r)$ contains the remaining diagonal mass.

In the code:

```
[U,Sv] = svd(H + H_prior, 'econ');
Vr     = U(:,1:k) * sqrt(Sv(1:k,1:k));
Dr     = diag(diag(H + H_prior) - sum(Vr.^2,2));
Dr     = max(Dr, 1e-8);

H_elbo = Vr*Vr' + diag(Dr);
```

so that

$$V_r = U_{(:,1:k)}S^{1/2}_{(1:k,1:k)}, \qquad D_r = \mathrm{diag}(H+H_{\mathrm{prior}}) - \sum_{j=1}^{k} V_{r,\cdot j}^2. \tag{41}$$

The update solves

$$H_{\mathrm{elbo}}\,\delta m = g, \qquad H_{\mathrm{elbo}} := V_r V_r^\top + \mathrm{diag}(D_r), \tag{42}$$

and updates

$$m \leftarrow m + \delta m \tag{43}$$

with an additional trust region and backtracking line search:

```
dm = solve_pd(H_elbo, g);
m_prev = m;
m      = m + dm;
```

The step is clipped if $\|\delta m\|$ exceeds a prescribed maximum.

8.3 ELBO approximation

Given H_{elbo} as the approximate posterior precision, the (Laplace) entropy term is

$$\mathcal{H}[q] = \frac{1}{2}\log|\Sigma_q| + \text{const} = -\frac{1}{2}\log|H_{\text{elbo}}| + \text{const}. \tag{44}$$

In the code:

```
Lchol   = chol(H_elbo,'lower');
logdetH = 2*sum(log(diag(Lchol)));
logL_entropy = -0.5 * logdetH;
```

The ELBO used for optimisation is then

$$\mathcal{F}(m) = \underbrace{\log p(R \mid m)}_{\text{logL\_lik}} + \underbrace{\log p(m)}_{\text{logL\_prior}} + \underbrace{\mathcal{H}[q]}_{\text{logL\_entropy (up to const)}}. \tag{45}$$

In code:

```
logL = logL_lik + logL_prior + logL_entropy;
```

A backtracking line search via `quick_elbo_gc` ensures non-decreasing ELBO, using a fast proxy that recomputes the GC residuals and likelihood but omits the entropy term.

9 Summary: What "Working in Generalised Coordinates" Means Here

In summary, the routine `fitVariationalLaplaceThermo_GC.m` replaces the standard VL ingredients

$$y,\ J_0$$

with their generalised-coordinate counterparts

$$R = \begin{bmatrix} y - \hat{y} \\ D(y-\hat{y}) \\ \vdots \\ D^q(y-\hat{y}) \end{bmatrix}, \qquad J_{\text{GC}} = \begin{bmatrix} J_0 \\ DJ_0 \\ \vdots \\ D^q J_0 \end{bmatrix}, \tag{46}$$

and augments the likelihood with:

- A Sobolev-type metric Γ that controls the relative importance of different derivative orders.
- A heteroscedastic, t-like noise model in GC space.
- A smoothness-regularised Gaussian prior $p(m) = \mathcal{N}(m; m_0, S_0 + K)$.
- A low-rank + diagonal posterior precision approximation for scalable ELBO estimation.

Operationally, "working in generalised coordinates" here means that all of the VL machinery (heteroscedastic noise, Gauss–Newton curvature, low-rank precision, entropy) is applied *after* embedding both residuals and Jacobians into GC space via the finite-difference operators D^k and the Sobolev metric Γ. This is exactly what the implementation of `fitVariationalLaplaceThermo_GC.m` realises.