Optimization algorithms illustrated using Matlab

Carl Tape, Qinya Liu, Albert Tarantola

June 1, 2006 (California Institute of Technology)

Contact: `ctape@alaska.alaska.edu`

Last compiled: July 14, 2019

Note

This set of notes complements the Matlab code `run_genlsq.m`, which is based on the optimization algorithms presented in *Tarantola* (2005, Section 6.22). The purpose of these notes and associated code is to gain some familiarity with several standard optimization algorithms in the context of generalized least squares (e.g., *Tarantola and Valette*, 1982).

These algorithms are using a simple ray-based traveltime measurements at stations for an earthquake described by its epicenter (x_s, y_s) and origin time (t_s) within a homogeneous medium with velocity v. The problem is outlined in *Tarantola* (2009, Section 4.4.2), where it is illustrated using Mathematica and a single optimizaton method; here we use Matlab and several different optimization methods.

Contents

1 The forward problem **3**

2 Set-up for the inverse problem **3**
2.1 Generating Gaussian random vectors . . . 3
2.2 Generating samples from a covariance matrix . . . 3
2.3 Specifying the prior model covariance . . . 3
2.4 Sampling the prior model to obtain a target model . . . 4
2.5 Specifying the data covariance . . . 4
2.6 Sampling the data covariance to obtain errors for the target data . . . 5
2.7 Normalization factors for $\mathbf{C}_\mathrm{M}$ and $\mathbf{C}_\mathrm{D}$ . . . 5
2.8 The misfit function and its gradient . . . 5
2.9 The posterior model and covariance matrix . . . 6

3 Optimization methods **6**
3.1 Newton . . . 6
3.2 quasi-Newton . . . 7
3.3 steepest descent . . . 7
3.4 conjugate gradient . . . 8
3.4.1 conjugate gradient (polynomial line search) . . . 9
3.5 variable metric (matrix version) . . . 9
3.6 variable metric (vector version) . . . 10
3.7 square-root variable metric (SRVM) (matrix version) . . . 11
3.8 square-root variable metric (SRVM) (vector version) . . . 12

4 Sampling the posterior distribution of models **13**
4.1 Option 1: Sampling $\mathbf{C}_\mathrm{post}$ using its matrix square root . . . 13
4.2 Option 2: The square-root variable metric method . . . 13

5 Convergence results for each method **14**

6 Running the code **14**

6.1 Example 1: One run, one method (steepest descent) 14
6.2 Example 2: One run, six methods 18
6.3 Example 2b: One run, all variable metric methods 18
6.4 Example 3a: 10 different runs, 1 method (steepest descent) 19
6.5 Example 3b: 10 different runs, 1 method (steepest descent) 19
6.6 Example 4: 100 different runs, six methods 20

A Forward model functions (`forward_epicenter.m`) **21**

A.1 First derivatives of $g(\mathbf{m})$ 21
A.2 Second derivatives of $\mathbf{g}(\mathbf{m})$ 22
A.3 Second derivatives of $S(\mathbf{m})$ 23

References **24**

List of Figures

1 Example 1: Source-receiver geometry 25
2 Example 1: Samples of the prior model 26
3 Example 1: Samples of the data covariance matrix, I 27
4 Example 1: Samples of the data covariance matrix, II 28
5 Example 1: Convergence curve for steepest descent 29
6 Example 1: Samples of the posterior 30
7 Example 2a 31
8 Example 2a 32
9 Example 2a: 33
10 Example 2b: 34
11 Example 3a: Posterior distributions for steepest descent 35
12 Example 3a: Convergence curves for steepest descent 36
13 Example 3a: Posterior distributions for steepest descent 37
14 Example 3a: Posterior models for steepest descent 38
15 Example 3b: Convergence curves for steepest descent (no errors) 39
16 Example 3b: Posterior models for steepest descent (no errors) 40
17 Example 4: Convergence for five optimization methods, I 41
18 Example 4: Convergence for five optimization methods, II 42
19 Example 4: Convergence for five optimization methods, III 43

1 The forward problem

The unknown model vector comprises information describing the epicenter $(x_s,\ y_s)$, the origin time t_s, and the velocity of the homogeneous medium. The data are travel times computed assuming straight line ray paths. In Appendix A we list the Matlab codes for the forward problems.

2 Set-up for the inverse problem

The fundamental dimensions in the problem are the $M = 4$ (Cartesian) model parameters and the $N = 12$ observations. This means that the matrices of interest — the design matrix, matrix of partial derivatives, covariance matrices — are easily computable and invertible.

2.1 Generating Gaussian random vectors

We will need Gaussian random vectors to sample covariance matrices: the prior model, the data, and the posterior model.

2.2 Generating samples from a covariance matrix

As described in *Tarantola* (2005, p. 117), a sample from a prescribed covraiance matrix, $\mathbf{C}$ can be generated assuming its matrix-square-root, $\mathbf{L}$, can be obtained [1]:

$$\mathbf{C} = \mathbf{L}\mathbf{L}^t. \tag{1}$$

Let $\mathbf{w}$ represent a Gaussian vector with zero mean and unit covariance ($\mathbf{C}_{\mathrm{w}} = \mathbf{I}$), easily computed in Matlab (Section 2.1). Then a random realization of $\mathbf{C}$ is computed by

$$\mathbf{x} = \mathbf{L}\mathbf{w}. \tag{2}$$

Each new $\mathbf{w}$ therefore leads to a new $\mathbf{x}$.

2.3 Specifying the prior model covariance

The the prior model distribution is given by a mean and its covariance.

We specify a prior model ("mean") as

$$\mathbf{m} = \begin{bmatrix} x_s \\ y_s \\ t_s \\ v \end{bmatrix} = \begin{bmatrix} 35.0\text{ km} \\ 45.0\text{ km} \\ 16.0\text{ s} \\ 1.61 \end{bmatrix}, \tag{3}$$

where $v = \ln(V/V_0)$ is the logarithmic velocity (Appendix A).

We specify the prior model covariance as

$$\mathbf{C}_{\text{prior}} = \begin{bmatrix} (10.0)^2 & 0 & 0 & 0 \\ 0 & (10.0)^2 & 0 & 0 \\ 0 & 0 & (0.5)^2 & 0 \\ 0 & 0 & 0 & (0.2)^2 \end{bmatrix}. \tag{4}$$

[1] Typically a Cholesky decomposition would be used to the matrix composition. However, in *many* cases, especially for large-dimension matrixes, numerical instabilities arise.

2.4 Sampling the prior model to obtain a target model

In Figure 2 we show 1000 samples of the prior model distribution, generated via the approach in Section 2.2. We randomly pick one to be the target model, $\mathbf{m}_{\text{target}}$. The Matlab code is given by

```
% prior model covariance matrix (assumed to be diagonal)
sigma_prior = [10 10 0.5 0.2]';         % standard deviations
cprior0     = diag( sigma_prior.^2 );   % diagonal covariance matrix
if inormalization==1
    Cmfac = nparm;
else
    Cmfac = 1;
end
cprior   = Cmfac * cprior0;             % WITH NORMALIZATION FACTOR
icprior  = inv(cprior);                 % WITH NORMALIZATION FACTOR
icprior0 = inv(cprior0);
Lprior   = chol(cprior0,'lower')';      % square-root (lower triangular)

% sample the prior model distribution using the square-root UNNORMALIZED covariance matrix
for xx=1:nsamples, randn_vecs_m(:,xx) = randn(nparm,1); end
cov_samples_m  = Lprior * randn_vecs_m;
mprior_samples = repmat(mprior,1,nsamples) + cov_samples_m;
```

where `randn_vecs_m` is a set of Gaussian random vectors, and `chol` is the Matlab function that performs a Cholesky decomposition of the prior model covariance matrix.

2.5 Specifying the data covariance

Similar to Section 2.3, the data are described in terms of the "target data" and the data covariance. We specify a data covariance matrix of

$$\mathbf{C}_\mathrm{D} = \begin{bmatrix} (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0.2)^2 \end{bmatrix}. \tag{5}$$

The exact "target data" are simply computed from the target model, that is

$$\mathbf{d}_{\text{target}} = \mathbf{g}\left(\mathbf{m}_{\text{target}}\right), \tag{6}$$

where $\mathbf{g}(\cdot)$ denotes the forward model. For each inversion, we add data errors associated with $\mathbf{C}_\mathrm{D}$ to the exact target data, i.e.,

$$\mathbf{d}_{\text{obs}} = \mathbf{d}_{\text{target}} + \boldsymbol{\epsilon}, \tag{7}$$

where $\boldsymbol{\epsilon}$ represents a Gaussian random sample of $\mathbf{C}_\mathrm{D}$ (Eq. 5).

2.6 Sampling the data covariance to obtain errors for the target data

Similar to Section 2.4, we now sample the data covariance to obtain the "data".

This process is shown in Figures 3 and 4, and the pertinent Matlab code is

```
% data covariance matrix (assumed to be diagonal)
tsigma = 0.5;                        % uncertainty in arrival time measurement, seconds
sigma_obs = tsigma * ones(ndata,1); % standard deviations
cobs0     = diag( sigma_obs.^2 );   % diagonal covariance matrix
if inormalization==1
    Cdfac = ndata;
else
    Cdfac = 1;
end
cobs   = Cdfac * cobs0;              % WITH NORMALIZATION FACTOR
icobs  = inv(cobs);                  % WITH NORMALIZATION FACTOR
icobs0 = inv(cobs0);
Lcobs  = chol(cobs0,'lower')';       % square-root (lower triangular)

% sample the data distribution using the square-root UNNORMALIZED covariance matrix
for xx=1:nsamples, randn_vecs_d(:,xx) = randn(ndata,1); end
cov_samples_d = Lcobs * randn_vecs_d;
dobs_samples  = repmat(dtarget,1,nsamples) + cov_samples_d;
```

Note that the samples are generated using the matrix described `cobs0`, not using the normalized version `cobs`; this distinction is discussed next.

2.7 Normalization factors for $\mathbf{C}_\mathrm{M}$ and $\mathbf{C}_\mathrm{D}$

In many problems, there are different classes of model or data parameters which one may wish to weight differently in computing a single value of the misfit function. Such weights provide have the effect of "balancing" the different parts of the misfit function. In the simplest case, one typically would like the norm operation of a vector ($\mathbf{m}$ or $\mathbf{d}$) to be approximately one, e.g., $\mathbf{d}^t\mathbf{C}_\mathrm{D}^{-1}\mathbf{d} \approx 1$. To achieve this, one simply incorporates the number of data as a factor in $\mathbf{C}_\mathrm{D}^{-1}$, as shown in the code excerpt in Section 2.6. Thus, we write the weighted version of the data covariance as

$$\mathbf{C}'_\mathrm{D} = N\mathbf{C}_\mathrm{D}, \tag{8}$$

where N is the total number of measurements, i.e., the number of entries in vector $\mathbf{d}$.

2.8 The misfit function and its gradient

The misfit function, $S(\mathbf{m})$, will depend on the choice of norm for the data space and for the model space. The nonlinear least-squares misfit function employs an L_2-norm in both data space and model space, and it is defined by (*Tarantola*, 2005, Eq. 6.251)

$$\begin{aligned} 2\,S(\mathbf{m}) &= \|\mathbf{g}(\mathbf{m}) - \mathbf{d}_\mathrm{obs}\|_\mathrm{D}^2 + \|\mathbf{m} - \mathbf{m}_\mathrm{prior}\|_\mathrm{M}^2 \\ &= (\mathbf{g}(\mathbf{m}) - \mathbf{d}_\mathrm{obs})^t\ \mathbf{C}_\mathrm{D}^{-1}\,(\mathbf{g}(\mathbf{m}) - \mathbf{d}_\mathrm{obs})\ +\ (\mathbf{m} - \mathbf{m}_\mathrm{prior})^t\ \mathbf{C}_\mathrm{prior}^{-1}\,(\mathbf{m} - \mathbf{m}_\mathrm{prior}) \end{aligned} \tag{9}$$

Here are two equivalent ways to implement the misfit function:

```
if 1==1
    % data misfit
    Sd = @(m,dobs,icobs) ( 0.5* ...
          (d(m)-dobs)' * icobs * (d(m)-dobs) );
    % model misfit (related to regularization)
    Sm = @(m,mprior,icprior) ( 0.5* ...
          (m - mprior)' * icprior * (m - mprior) );
    % total misfit
    S = @(m,dobs,mprior,icobs,icprior) ( Sd(m,dobs,icobs) + Sm(m,mprior,icprior) );

else
    S = @(m,dobs,mprior,icobs,icprior) ( 0.5*( ...
          (d(m)-dobs)' * icobs * (d(m)-dobs) ...
        + (m - mprior)' * icprior * (m - mprior) ) );
end
```

2.9 The posterior model and covariance matrix

```
% posterior covariance matrix (e.g., Tarantola Eq. 3.53)
% note: cpost0 does not include normalization factors Cdfac and Cmfac
Gpost = G(mpost);
cpost0 = inv(Gpost'*icobs0*Gpost + icprior0);
```

3 Optimization methods

3.1 Newton

```
for nn = 1:niter
    disp([' iteration ' num2str(nn) ' out of ' num2str(niter) ]);
    m     = mnew;
    dpred = d(m);
    Ga    = G(m);

    % update the model: Tarantola (2005), Eq 6.319
    % (the line-search parameter is assumed to be nu = 1)
    ghat  = Ga'*icobs*(dpred - dobs) + icprior*(m - mprior);  % gradient
    Hhat1 = icprior + Ga'*icobs*Ga;                            % approximate Hessian
    Hhat2 = zeros(nparm,nparm);
    % The ONLY difference between quasi-Newton and Newton is the
    % Hhat2 term. The iith entry of the residual vector is the
    % weight for the corresponding matrix of partial derivatives (G2).
    % Note that the observations are present in Hhat2 but not in Hhat1.
    dwt = icobs*(dpred-dobs);
    for ii=1:ndata
        Hhat2 = dwt(ii) * G2(m,ii);
    end
    Hhat  = Hhat1 + Hhat2;                                    % full Hessian
    disp('Hhat = Hhat1 + Hhat2:');
    for kk=1:nparm
        disp(sprintf('%8.4f %8.4f %8.4f %8.4f   %8.4f %8.4f %8.4f %8.4f + %8.4f %8.4f %8.4f %8.4f',...
            Hhat(kk,:),Hhat1(kk,:),Hhat2(kk,:)));
    end

    if 1==1
        %mnew  = m - inv(Hhat)*ghat;
        dm    = -Hhat\ghat;
        mnew  = m + dm;
    else
        % equivalent formula: see Tarantola and Valette (1982), Eq. 23-35
        mutemp = (Ga*cprior*Ga' + cobs) \ (dobs-dpred + Ga*(m - mprior));
        mnew  = mprior + cprior*Ga'*mutemp;
    end

    % misfit function for new model
    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
end
```

3.2 quasi-Newton

```
for nn = 1:niter
    disp([' iteration ' num2str(nn) ' out of ' num2str(niter) ]);
    m     = mnew;
    dpred = d(m);
    Ga    = G(m);

    % update the model: Tarantola (2005), Eq 6.319
    % (the line-search parameter is assumed to be nu = 1)
    Hhat  = icprior + Ga'*icobs*Ga;                           % approximate Hessian
    ghat  = Ga'*icobs*(dpred - dobs) + icprior*(m - mprior);  % gradient

    if 1==1
        %mnew  = m - inv(Hhat)*ghat;
        dm    = -Hhat\ghat;
        mnew  = m + dm;
    else
        % equivalent formula: see Tarantola and Valette (1982), Eq. 23-35
        mutemp = (Ga*cprior*Ga' + cobs) \ (dobs-dpred + Ga*(m - mprior));
        mnew  = mprior + cprior*Ga'*mutemp;
    end

    % misfit function for new model
    % note: book-keeping only -- not used within the algorithm above
    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
end
```

3.3 steepest descent

```
% NOTE: Use Eq. 6.315 as a preconditioner to convert the
%       preconditioned steepest descent to the quasi-Newton method.
F = F0; % identity
for nn = 1:niter
    disp([' iteration ' num2str(nn) ' out of ' num2str(niter) ]);
    m = mnew;

    % steepest ascent vector (Eq. 6.307 or 6.312)
    dpred = d(m);
    Ga    = G(m);
    g     = cprior*Ga'*icobs*(dpred - dobs) + (m - mprior);
    % search direction (preconditioned by F) (Eq. 6.311)
    p = F*g;

    % update the model
    b    = Ga*p;
    mu   = g'*icprior*p / (p'*icprior*p + b'*icobs*b);  % Eq. 6.314 (Eq. 6.309 if F = I)
    mnew = m - mu*p;    % Eq 6.297

    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
end
```

3.4 conjugate gradient

```
for nn = 1:niter
    disp([' iteration ' num2str(nn) ' out of ' num2str(niter) ]);
    m = mnew;

    % steepest ascent vector (Tarantola, 2005, Eq. 6.312)
    dpred = d(m);
    Ga    = G(m);
    g     = cprior*Ga'*icobs*(dpred - dobs) + (m - mprior);

    % search direction (Tarantola, 2005, Eq. 6.329)
    l = F0*g;
    if nn == 1
       alpha = 0; p = g;
    else
       alpha = (g-gold)'*icprior*l / (gold'*icprior*lold);  % Eq. 6.331-2
       p = l + alpha*pold;
    end

    % calculate step length
    b = Ga*p;
    mu = g'*icprior*p / (p'*icprior*p + b'*icobs*b);  % Eq. 6.333

    % update model
    mnew = m - mu*p;
    gold = g;
    pold = p;
    lold = l;

    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
end
```

3.4.1 conjugate gradient (polynomial line search)

Here, instead of using the line search suggested by *Tarantola* (2005, Eq. 6.333), we use the line search outlined in *Tape et al.* (2007). The main idea is that you compute the misfit function one more time (per iteration), which allows you to pick a better step length. The code for this line search is as follows:

```
% test model using quadratic extrapolation: Tape-Liu-Tromp (2007)
% (compared with the previous CG algorithm, we do not use b = Ga*b to get mu)
mu_test   = -2*Sval / sum( g'*icprior*p );
m_test    = m + mu_test*p;
Sval_test = S(m_test,dobs,mprior,icobs,icprior);

% end iteration if the test model is unrealistic
if ~isreal(Sval_test)
    disp('polynomial step is TOO FAR');
    S_vec(nn+1:end) = S_vec(nn);
    break
end

% determine coefficients of quadratic polynomial (ax^2 + bx + c),
% using the two points and one slope
x1 = 0;
x2 = mu_test;
y1 = Sval;
y2 = Sval_test;
g1 = sum( g'*icprior*p );
Pa = ((y2 - y1) - g1*(x2 - x1)) / (x2^2 - x1^2);
Pb = g1;
Pc = y1 - Pa*x1^2 - Pb*x1;

% get the mu value associated with analytical minimum of the parabola
if Pa ~= 0, mu = -Pb / (2*Pa); else error('check the input polynomial'); end
```

3.5 variable metric (matrix version)

```
F = F0;
for nn = 1:niter
    disp([' iteration ' num2str(nn) ' out of ' num2str(niter) ]);
    m = mnew;

    % steepest ascent vector
    dpred = d(m);
    Ga    = G(m);
    g     = cprior*Ga'*icobs*(dpred - dobs) + (m - mprior);

    % update the preconditioner F
    if nn > 1
        dg = g - gold;                             % Eq. 6.341
        v  = F*dg;                                 % Eq. 6.355
        u  = dm - v;                               % Eq. 6.341
        F  = F + u*u'*icprior / (u'*icprior*dg);   % Eq. 6.356
    end

    % preconditioning search direction (Eq. 6.355)
    p = F*g;
    % update the model
    b    = Ga*p;                                        % Eq. 6.333
    mu   = g'*icprior*p / (p'*icprior*p + b'*icobs*b);  % Eq. 6.333
    dm   = -mu*p;                                       % Eq. 6.355
    mnew = m + dm;
    gold = g;

    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
end

% estimated posterior covariance matrix from F_hat, Eq. 6.362
% compare with cpost = inv(Gpost'*icobs*Gpost + icprior)
Fhat = F * cprior
```

3.6 variable metric (vector version)

The vector-based version of the variable metric algorithm is particularly well-suited for problems in which the model dimension is very large — say, M is on the order of millions — in which case, it is not feasible to either store $M \times M$ matrices or to perform operations with such a matrix. However, it is possible to store a set of scalars and vectors so that an *operation* by such a matrix on an arbitrary vector $\boldsymbol{\chi}$ is possible (*Tarantola*, 2005, Section 6.22.8).

```
u = NaN(nparm,niter-1);
v = NaN(nparm,niter-1);
for nn = 1:niter
    m = mnew;

    % steepest ascent vector
    delta = d(m);
    Ga    = G(m);
    g     = cprior*Ga'*icobs*(delta - dobs) + (m - mprior);

    % update the preconditioner F (Section 6.22.8)
    if nn > 1
        dg = g - gold;
        F_dg = F0*dg;
        % compute u(k) and v(k)
        for jj = 1:(nn-2)  % loop is entered only if nn >= 3
            vtmp = dg'*icprior*u(:,jj);
            F_dg = F_dg + vtmp/v(jj) * u(:,jj);     % Eq. 6.347
        end
        u(:,nn-1) = dm - F_dg;                      % Eq. 6.341
        v(nn-1) = dg'*icprior*u(:,nn-1);            % Eq. 6.348
    end

    % preconditioning search direction p = F_g (Eq. 6.340, Eq. 6.347)
    p = F0*g;
    for jj = 1:(nn-1)   % loop is entered only if nn >= 2
        p = p + g'*icprior*u(:,jj)/v(jj) * u(:,jj);
    end

    % update the model
    b    = Ga*p;
    mu   = g'*icprior*p / (p'*icprior*p + b'*icobs*b);  % Eq. 6.333
    dm   = -mu*p;
    mnew = m + dm;
    gold = g;

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);
end

% estimated posterior covariance matrix from F_hat, Eq. 6.362
F = vm_F(F0,icprior,u,v);
Fhat = F * cprior
```

The vector version of the variable metric method makes use of `vm_F.m` and `vm_F_chi.m`.

3.7 square-root variable metric (SRVM) (matrix version)

The square-root variable metric (SRVM) is described in *Williamson* (1975) and *Hull and Tapley* (1977).

```
% We should have F = Shat * Shat' * icprior
F = F0;
F0hat = F0*cprior;
S0hat = sqrtm(F0hat);  % works if Fhat is symmetric positive definite
% check: a0 = rand(2,2); a = a0*a0', b = sqrtm(a), b*b
Shat = S0hat;
Fhat = F0hat;

% store these for checking only
a_vec  = zeros(niter-1,1);
b_vec  = zeros(niter-1,1);
nu_vec = zeros(niter-1,1);
w_mat  = zeros(nparm,niter-1);

for nn = 1:niter
    m = mnew;

    % gradient
    delta = d(m);
    Ga    = G(m);
    ghat  = Ga'*icobs*(delta - dobs) + icprior*(m - mprior);

    % update the preconditioner F (using S); see Hull and Tapley (1977)
    if nn >= 2
        dghat = ghat - ghat_old;
        yhat  = mu*ghat_old + dghat;
        w     = Shat'*yhat;
        a     = yhat'*Fhat*dghat;
        b     = w'*w;
        nu    = srvm_nu(a,b);
        Shat  = Shat*(eye(nparm) - nu/a*w*w' );
        Fhat  = Shat*Shat';

        % for checking only
        a_vec(nn-1) = a;
        b_vec(nn-1) = b;
        nu_vec(nn-1) = nu;
        w_mat(:,nn-1) = w;
    end

    % preconditioned gradient
    p = Fhat*ghat;

    % update the model
    c    = Ga*p;
    mu   = ghat'*p / (p'*icprior*p + c'*icobs*c);  % Eq. 6.333
    %mu   = g'*icprior*p / (p'*icprior*p + c'*icobs*c);  % Eq. 6.333
    dm   = -mu*p;
    mnew = m + dm;
    ghat_old = ghat;

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);
end

% estimated posterior covariance matrix from F_hat, Eq. 6.362
Fhat

% estimated posterior covariance matrix computed from stored vectors and scalars
Fhat_check = srvm_Fhat(S0hat,niter-1,nu_vec,a_vec,w_mat)
a = a_vec;
b = b_vec;
nu = nu_vec;
w = w_mat;
```

3.8 square-root variable metric (SRVM) (vector version)

The motivation for the vector-based version of SRVM is the same as described in Section 3.6, only in this case, the operation $\hat{\mathbf{S}}\boldsymbol{\chi}$ will generate a sample of the posterior model distribution, where $\boldsymbol{\chi}$ is a Gaussian random vector. The key function for this algorithm is `srvm_Shat_chi.m`.

For application to seismic imaging, see *Luo* (2012), *Liu et al.* (2019), *Liu and Peter* (2019).

```
% We should have F = Shat * Shat' * icprior
F = F0;
F0hat = F0*cprior;
S0hat = sqrtm(F0hat);
Shat = S0hat;

% initialize vectors and scalars
a  = zeros(niter-1,1);
b  = zeros(niter-1,1);
nu = zeros(niter-1,1);
w  = zeros(nparm, niter-1);

for nn = 1:niter
    m = mnew;

    % gradient
    delta = d(m);
    Ga    = G(m);
    ghat  = Ga'*icobs*(delta - dobs) + icprior*(m - mprior);

    % update the preconditioner F (using S)
    % see Hull and Tapley (1977)
    if nn >= 2
        dghat = ghat - ghat_old;
        yhat  = mu*ghat_old + dghat;

        w(:,nn-1) = srvm_Shat_chi(yhat,nn-2,S0hat,nu,a,w,1);
        beta      = w(:,nn-1) - mu*ShatT_ghat;
        a(nn-1)   = transpose(w(:,nn-1))*beta(:);
        b(nn-1)   = transpose(w(:,nn-1))*w(:,nn-1);
        nu(nn-1)  = srvm_nu(a(nn-1),b(nn-1));
    end

    % update the search direction (does nothing for nn=1)
    ShatT_ghat = srvm_Shat_chi(ghat,nn-1,S0hat,nu,a,w,1);
    p          = srvm_Shat_chi(ShatT_ghat,nn-1,S0hat,nu,a,w,0);

    % update the model
    c        = Ga*p;
    mu       = ghat'*p / (p'*icprior*p + c'*icobs*c);  % Eq. 6.333
    %mu       = g'*icprior*p / (p'*icprior*p + c'*icobs*c);  % Eq. 6.333
    dm       = -mu*p;
    mnew     = m + dm;
    ghat_old = ghat;

    disp(sprintf('%i/%i : prior, current, target:',nn,niter));
    disp([mprior mnew mtarget]);
    Sd_vec(nn+1) = Sd(mnew,dobs,icobs);
    Sm_vec(nn+1) = Sm(mnew,mprior,icprior);
    S_vec(nn+1) = S(mnew,dobs,mprior,icobs,icprior);
end

% estimated posterior covariance matrix computed from stored vectors and scalars
Fhat = srvm_Fhat(S0hat,niter-1,nu,a,w)
```

4 Sampling the posterior distribution of models

4.1 Option 1: Sampling $\mathbf{C}_{\text{post}}$ using its matrix square root

If it is possible to directly compute the square root of the posterior covariance matrix, then Equation (2) can be used to generate samples of $\mathbf{C}_{\text{post}}$.

4.2 Option 2: The square-root variable metric method

The main advantage of the square-root variable metric method is that we are able to sample the posterior model distribution. The code to sample the posterior is similar to that in Section 2.4 and Section 2.6, only in this case we do not have any matrix stored, but rather a set of vectors and scalars that can compute the vector $\hat{\mathbf{S}}\boldsymbol{\chi}$.

```
mcov_samples = zeros(nparm,nsamples);
for ii=1:nsamples, randn_vecs_m(:,ii) = randn(nparm,1); end
for kk = 1:nsamples
    chi = randn_vecs_m(:,kk);
    mcov_samples(:,kk) = srvm_Shat_chi(chi,niter-1,S0hat,nu,a,w,0);
end
mpost_samples = repmat(mpost,1,nsamples) + mcov_samples;
```

5 Convergence results for each method

Using the misfit function in Equation (9), we can plot $S_k = S(\mathbf{m}_k)$ for a particular method, as shown in Figure 5. Each point represents the misfit function for the mean model.

We also have the option of choosing dozens of different target models from the prior distribution, and then comparing the convergence curves. The "mean curve" (the mean of all the convergence curves) gives some idea of the general shape of the convergence. This reduces the chance that the shape of the convergence curve has characteristics that result from a particular (randomly sampled) choice of the target model.

The epicenter problem is fast, and we can compare all methods for hundreds of sets of "observations" generated from different target models (with appropriate errors added). Figures 17 and 18 show a collection of convergence curves for six different methods. The "plateau" that typically appears in the variable metric method between iterations 2 and 4 may be related to an improper choice of line search.

6 Running the code

The source–receiver geometry for the test problem is shown in Figure 1.

6.1 Example 1: One run, one method (steepest descent)

To execute the code, type `run_genlsq` in the Matlab command window. This should lead to the following prompt (type in the entries, as indicated below):

```
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
Optimization problem set-up:
Forward problem (1 = epicenter; 2 = epicenter-cresent): 1
Select the number of samples for the distributions (1000): 1000
Type 1 for random initial model or 0 for fixed: 0
Type 1 for random target model or 0 for fixed: 0
Data errors (0 = none, 1 = random, 2 = fixed): 2
Type 1 to plot figures or 0 to not: 1
```

Here we have entered options for 1000 samples of the distributions, a fixed initial model, a fixed target model, and fixed errors added to the target data. The user is encouraged to examine the subsequent figures before proceeding. Figure 2 shows two representations of the 1000 4-parameter samples, in addition to the initial model ($\mathbf{m}_{00}$) and target model ($\mathbf{m}_{\text{target}}$) for this example. Figures 3 and 4 show samples of the data for all 1000 models.

Next, the user is faced with the following prompt:

```
Optimization methods:
    0 : none (stop here)
    1 : Newton
    2 : quasi-Newton
    3 : steepest descent
    4 : conjugate gradient
    5 : conjugate gradient (polynomial line search)
    6 : variable metric (matrix version)
    7 : variable metric (vector version)
    8 : square-root variable metric (matrix version)
    9 : square-root variable metric (vector version)
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
   IF YOU WANT MULTIPLE METHODS, LIST THE NUMBERS IN BRACKETS (e.g., [1 4 5])
   IF YOU WANT ALL METHODS, USE [1:9]
Select your optimization method(s) (1-9): 3
Select the number of iterations (10): 10
```

Here we have entered options for the steepest descent method with 10 iterations (for one run). The code outputs the following:

```
 Initial model 1 out of 1
==========================================================
Running steepest descent...
==========================================================
 iteration 0 out of 10
0/10 : prior, current, target:
   35.0000   46.5236   21.2922
   45.0000   40.1182   46.2974
   16.0000   15.3890   16.1314
    1.6094    1.7748    2.0903
 iteration 1 out of 10
1/10 : prior, current, target:
   35.0000   32.5197   21.2922
   45.0000   46.0045   46.2974
   16.0000   15.3494   16.1314
    1.6094    1.9069    2.0903
 iteration 2 out of 10
2/10 : prior, current, target:
   35.0000   26.4517   21.2922
   45.0000   45.1591   46.2974
   16.0000   15.4300   16.1314
    1.6094    1.8444    2.0903
 iteration 3 out of 10
3/10 : prior, current, target:
   35.0000   25.1558   21.2922
   45.0000   46.5218   46.2974
   16.0000   15.3991   16.1314
    1.6094    1.9042    2.0903
 iteration 4 out of 10
4/10 : prior, current, target:
   35.0000   23.2082   21.2922
   45.0000   46.1433   46.2974
   16.0000   15.4238   16.1314
    1.6094    1.8949    2.0903
 iteration 5 out of 10
5/10 : prior, current, target:
   35.0000   22.8829   21.2922
   45.0000   46.3288   46.2974
   16.0000   15.4184   16.1314
    1.6094    1.9225    2.0903
 iteration 6 out of 10
6/10 : prior, current, target:
   35.0000   21.9929   21.2922
   45.0000   46.0784   46.2974
   16.0000   15.4378   16.1314
    1.6094    1.9194    2.0903
 iteration 7 out of 10
7/10 : prior, current, target:
   35.0000   21.9021   21.2922
   45.0000   46.1236   46.2974
   16.0000   15.4418   16.1314
    1.6094    1.9349    2.0903
 iteration 8 out of 10
8/10 : prior, current, target:
   35.0000   21.4170   21.2922
   45.0000   45.9621   46.2974
   16.0000   15.4597   16.1314
    1.6094    1.9331    2.0903
 iteration 9 out of 10
9/10 : prior, current, target:
   35.0000   21.4273   21.2922
   45.0000   45.9958   46.2974
   16.0000   15.4671   16.1314
    1.6094    1.9435    2.0903
 iteration 10 out of 10
10/10 : prior, current, target:
   35.0000   21.1243   21.2922
   45.0000   45.8870   46.2974
   16.0000   15.4839   16.1314
    1.6094    1.9418    2.0903
 iteration 0 out of 10
0/10 : prior, current, target:
   35.0000   21.1243   21.2922
   45.0000   45.8870   46.2974
   16.0000   15.4839   16.1314
    1.6094    1.9418    2.0903
```

The convergence curve is shown in Figure 5.

Because the problem is small, we can compute $\mathbf{C}_{\mathrm{post}}$ no matter what optimization method is selected. By taking the square root of $\mathbf{C}_{\mathrm{post}}$, we can generate samples of the posterior distribution, as shown in Figure 6. Note that a gradient-based optimization does *not*, in general, provide $\mathbf{C}_{\mathrm{post}}$.

The following output summarizes the iterative inversion:

```
cpost0 =
    4.0852    0.5191   -0.0868   -0.0589
    0.5191    2.2696   -0.0128   -0.0169
   -0.0868   -0.0128    0.0868    0.0129
   -0.0589   -0.0169    0.0129    0.0029
rho_post =
    1.0000    0.1705   -0.1457   -0.5367
    0.1705    1.0000   -0.0287   -0.2073
   -0.1457   -0.0287    1.0000    0.8058
   -0.5367   -0.2073    0.8058    1.0000
model summary (10 iterations):
    prior    initial  posterior   target
   35.0000   46.5236   21.1243   21.2922
   45.0000   40.1182   45.8870   46.2974
   16.0000   15.3890   15.4839   16.1314
    1.6094    1.7748    1.9418    2.0903
data summary (12 observations):
    prior    initial   posterior   target   actual
   23.0711   22.4575   19.5256   19.6702   18.8013
   21.3852   22.0746   17.5467   17.8942   17.4276
   26.2956   25.8692   22.0098   21.7127   21.6611
   21.0111   19.4670   19.5895   19.7077   19.9488
   18.0276   18.7600   17.6693   17.9684   18.2509
```

```
  25.0062   24.1355   22.0496   21.7366   21.5065
  22.6165   19.2083   21.7912   21.5817   21.7794
  20.7726   18.4420   20.7472   20.6359   20.9650
  25.9889   24.0179   23.6100   23.0836   24.0899
  26.2956   22.0098   24.7096   24.0856   24.6530
  25.2195   21.5993   24.0299   23.4699   23.6047
  28.7279   25.5726   26.0369   25.1811   25.6414

Compare model uncertainties :
            model parameter :             xs            ys            ts             v
                      units :             km            km             s          none
                sigma_prior =       10.00000      10.00000       0.50000       0.20000
                 sigma_post =        2.02118       1.50652       0.29469       0.05428
  std(  1000 mpost_samples) =        2.00323       1.50464       0.29708       0.05420
   sigma_post / sigma_prior =        0.20212       0.15065       0.58937       0.27139

   iter              Sd              Sm     S = Sm + Sd
      0   14.0113335953    0.4678940978   14.4792276931
      1    3.1088570163    0.4971076295    3.6059646457
      2    1.3534389282    0.4263691881    1.7798081163
      3    0.7835111960    0.5760238099    1.3595350059
      4    0.6091460104    0.5960049914    1.2051510018
      5    0.4791315017    0.6610991518    1.1402306535
      6    0.4353347434    0.6712274803    1.1065622237
      7    0.3847483631    0.7029226432    1.0876710063
      8    0.3702321343    0.7051689856    1.0754011199
      9    0.3445445947    0.7222710148    1.0668156095
     10    0.3401552891    0.7200477216    1.0602030107
```

6.2 Example 2: One run, six methods

We now run all of the optimization algorithms for the same run as in Example 1. For this example, enter the following options:

```
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
Optimization problem set-up:
Forward problem (1 = epicenter; 2 = epicenter-cresent): 1
Select the number of samples for the distributions (1000): 1000
Type 1 for random initial model or 0 for fixed: 0
Type 1 for random target model or 0 for fixed: 0
Data errors (0 = none, 1 = random, 2 = fixed): 2
Type 1 to plot figures or 0 to not: 1

Optimization methods:
    0 : none (stop here)
    1 : Newton
    2 : quasi-Newton
    3 : steepest descent
    4 : conjugate gradient
    5 : conjugate gradient (polynomial line search)
    6 : variable metric (matrix version)
    7 : variable metric (vector version)
    8 : square-root variable metric (matrix version)
    9 : square-root variable metric (vector version)
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
   IF YOU WANT MULTIPLE METHODS, LIST THE NUMBERS IN BRACKETS (e.g., [1 4 5])
   IF YOU WANT ALL METHODS, USE [1:9]
Select your optimization method(s) (1-9): [1:6]
Select the number of iterations (10): 10
```

Here, each convergence algorithm will use the same initial model, target model, and data errors. Figure 7 and Figure 8 compares with convergence curves. Figure 9 shows the posterior distributions of epicenters.

6.3 Example 2b: One run, all variable metric methods

We now check that the convergence for all four variable metric algorithms is identical.

```
Select your optimization method(s) (1-9): [6:9]
```

Figure 10 shows the equivalent convergence curves, as expected.

The main difference is in how the posterior samples are generated. For the first seven algorithms, the samples are generated from $\mathbf{C}_{\text{post}}$ via Equation (2). For the square-root variable metric algorithm (algorithms 8 and 9), the samples of $\mathbf{C}_{\text{post}}$ are generated using `srvm_Shat_chi.m`.

6.4 Example 3a: 10 different runs, 1 method (steepest descent)

We now consider 10 different runs, with each run characterized by a randomly selected initial model but with a fixed target model. Furthermore, for each run we add a different set errors to the target data to generate the actual data. For this example, enter the following options:

```
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
Optimization problem set-up:
Forward problem (1 = epicenter; 2 = epicenter-cresent): 1
Select the number of samples for the distributions (1000): 1000
Type 1 for random initial model or 0 for fixed: 1
Type 1 for random target model or 0 for fixed: 0
Data errors (0 = none, 1 = random, 2 = fixed): 1
Type 1 to plot figures or 0 to not: 1

Optimization methods:
    0 : none (stop here)
    1 : Newton
    2 : quasi-Newton
    3 : steepest descent
    4 : conjugate gradient
    5 : conjugate gradient (polynomial line search)
    6 : variable metric (matrix version)
    7 : variable metric (vector version)
    8 : square-root variable metric (matrix version)
    9 : square-root variable metric (vector version)
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
   IF YOU WANT MULTIPLE METHODS, LIST THE NUMBERS IN BRACKETS (e.g., [1 4 5])
   IF YOU WANT ALL METHODS, USE [1:9]
Select your optimization method(s) (1-9): 3
Select the number of iterations (10): 10
Select the number of different runs for each inversion method (1 <= nrun <= 1000): 10
```

Figure 11 shows convergence curves and how they vary with different choices of initial model and data errors. Figure 12 is a collection of 10 convergence curves. Figure 13 represents the epicenter posterior distributions for six of the different runs. Figure 14 shows the initial and final models for the 10 different runs.

6.5 Example 3b: 10 different runs, 1 method (steepest descent)

For this example, the user should enter the same options as in Section 6.4, but with one difference:

```
Data errors (0 = none, 1 = random, 2 = fixed): 0
```

This allows us to examine the impact of using data errors that are based on our assumed data covariance matrix. (Note that the 10 initial models will be randomly chosen, so they will be different in Example 3b from in Example 3a.)

From Figures 15 and 16, we see that the data misfit term is lower for the errors-added case (3b), and there is no spread in the final epicenters (Figure 16 vs Figure 14).

6.6 Example 4: 100 different runs, six methods

For this example, the user should enter the following options:

```
 TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
 Optimization problem set-up:
 Forward problem (1 = epicenter; 2 = epicenter-cresent): 1
 Select the number of samples for the distributions (1000): 1000
 Type 1 for random initial model or 0 for fixed: 1
 Type 1 for random target model or 0 for fixed: 1
 Data errors (0 = none, 1 = random, 2 = fixed): 1
 Type 1 to plot figures or 0 to not: 0

Optimization methods:
    0 : none (stop here)
    1 : Newton (full Hessian)
    2 : quasi-Newton
    3 : steepest descent
    4 : conjugate gradient
    5 : conjugate gradient (polynomial line search)
    6 : variable metric (matrix version)
TYPE A NUMBER AFTER EACH PROMPT AND HIT ENTER:
   IF YOU WANT MULTIPLE METHODS, LIST THE NUMBERS IN BRACKETS (e.g., [1 4 5])
   IF YOU WANT ALL METHODS, USE [1:6]
Select your optimization method(s) (0-6): [1:6]
 Select the number of iterations (10): 10
 Select the number of different runs for each inversion method (1 <= nmodel <= 1000): 100
```

The output is shown in Figures 17 and 18. Figure 19 is the same as Figure 18, but for 1000 different runs of each method instead of 100.

A Forward model functions (`forward_epicenter.m`)

The forward model requires the following:

- Source location (x_s, y_s).
- Source origin time t_s.
- Homogeneous velocity of the medium V.
- Receiver location (x_r, y_r).

We define a logarithmic velocity, v, which is a Cartesian parameter. Transforming between v and V is given by

$$v = \ln(V/V_0) \quad (10)$$
$$V = V_0 \, \exp(v) \quad (11)$$

where V_0 is a scaling velocity (we use $V_0 = 1$ km/s).

The arrival time at the receiver is computed by integrating the slowness ($1/V$) along the ray path between source and receiver:

$$t_r = t_s + \int_{\text{ray}} \frac{1}{V(x,y)} \mathrm{d}s. \quad (12)$$

Assuming a spatially homogeneous velocity, $V(x,y) = V$, the ray paths are straight lines, and thus we obtain

$$t_r = t_s + \frac{\sqrt{(x_r - x_s)^2 + (y_r - y_s)^2}}{V}\,, \quad (13)$$

which can be written as

$$g(\mathbf{m}) = g(x_s, y_s, t_s, v) = t_s + \frac{\sqrt{(x_r - x_s)^2 + (y_r - y_s)^2}}{V_0 \, \exp(v)}\,, \quad (14)$$

where $g(\cdot)$ represents the forward model, $\mathbf{m}$ is the input model (Eq. 3), and the receiver locations are assumed to be known and fixed.

A.1 First derivatives of $g(\mathbf{m})$

The partial derivatives of the arrival-time function (Eq. 14) are

$$\frac{\partial g}{\partial x_s} = -\left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-1/2} (x_r - x_s) \; V_0^{-1} \; \exp(-v) \quad (15)$$
$$\frac{\partial g}{\partial y_s} = -\left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-1/2} (y_r - y_s) \; V_0^{-1} \; \exp(-v) \quad (16)$$
$$\frac{\partial g}{\partial t_s} = 1 \quad (17)$$
$$\frac{\partial g}{\partial v} = -\left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{1/2} \; V_0^{-1} \; \exp(-v) \quad (18)$$

We now discretize the problem. There are N arrival-time measurements in the data vector $\mathbf{d}$. We assume a single event with N receivers, and we use index i to denote the measurement index, which in our case is also the receiver index. Then the $N \times M$ matrix of partial derivatives is given by

$$\mathbf{G} = \begin{bmatrix} \frac{\partial g_1}{\partial x_s} & \frac{\partial g_1}{\partial y_s} & \frac{\partial g_1}{\partial t_s} & \frac{\partial g_1}{\partial v} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial g_i}{\partial x_s} & \frac{\partial g_i}{\partial y_s} & \frac{\partial g_i}{\partial t_s} & \frac{\partial g_i}{\partial v} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial g_N}{\partial x_s} & \frac{\partial g_N}{\partial y_s} & \frac{\partial g_N}{\partial t_s} & \frac{\partial g_N}{\partial v} \end{bmatrix}. \tag{19}$$

The subscript for g_i indicates that each prediction of the arrival time will depend on the location of the i receiver.

The Matlab code for these commands can be found in `forward_epicenter.m`.

A.2 Second derivatives of g(m)

The 16 second derivatives of the arrival time function (Eq. 14) are then

$$\frac{\partial^2 g}{\partial x_s \partial x_s} = \left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-3/2} (y_r - y_s)^2 \ V_0^{-1} \ \exp(-v) \tag{20}$$

$$\frac{\partial^2 g}{\partial y_s \partial x_s} = -\left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-3/2} (x_r - x_s)(y_r - y_s) \ V_0^{-1} \ \exp(-v) \tag{21}$$

$$\frac{\partial^2 g}{\partial t_s \partial x_s} = \frac{\partial^2 g}{\partial x_s \partial t_s} = 0 \tag{22}$$

$$\frac{\partial^2 g}{\partial v \partial x_s} = \left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-1/2} (x_r - x_s) \ V_0^{-1} \ \exp(-v) \tag{23}$$

$$\frac{\partial^2 g}{\partial x_s \partial y_s} = \frac{\partial^2 g}{\partial y_s \partial x_s} \tag{24}$$

$$\frac{\partial^2 g}{\partial y_s \partial y_s} = \left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-3/2} (x_r - x_s)^2 \ V_0^{-1} \ \exp(-v) \tag{25}$$

$$\frac{\partial^2 g}{\partial t_s \partial y_s} = \frac{\partial^2 g}{\partial y_s \partial t_s} = 0 \tag{26}$$

$$\frac{\partial^2 g}{\partial v \partial y_s} = \left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{-1/2} (y_r - y_s) \ V_0^{-1} \ \exp(-v) \tag{27}$$

$$\frac{\partial^2 g}{\partial x_s \partial t_s} = 0 \tag{28}$$

$$\frac{\partial^2 g}{\partial y_s \partial t_s} = 0 \tag{29}$$

$$\frac{\partial^2 g}{\partial t_s \partial t_s} = 0 \tag{30}$$

$$\frac{\partial^2 g}{\partial v \partial t_s} = 0 \tag{31}$$

$$\frac{\partial^2 g}{\partial x_s \partial v} = \frac{\partial^2 g}{\partial v \partial x_s} \tag{32}$$

$$\frac{\partial^2 g}{\partial y_s \partial v} = \frac{\partial^2 g}{\partial v \partial y_s} \tag{33}$$

$$\frac{\partial^2 g}{\partial t_s \partial v} = 0 \tag{34}$$

$$\frac{\partial^2 g}{\partial v \partial v} = \left[(x_r - x_s)^2 + (y_r - y_s)^2\right]^{1/2} \; V_0^{-1} \; \exp(-v) \tag{35}$$

The matrix of second derivatives is given by

$$\frac{\partial^2 g}{\partial \mathbf{m}^2} = \begin{bmatrix} \frac{\partial^2 g}{\partial x_s \partial x_s} & \frac{\partial^2 g}{\partial y_s \partial x_s} & \frac{\partial^2 g}{\partial t_s \partial x_s} & \frac{\partial^2 g}{\partial v \partial x_s} \\ \frac{\partial^2 g}{\partial x_s \partial y_s} & \frac{\partial^2 g}{\partial y_s \partial y_s} & \frac{\partial^2 g}{\partial t_s \partial y_s} & \frac{\partial^2 g}{\partial v \partial y_s} \\ \frac{\partial^2 g}{\partial x_s \partial t_s} & \frac{\partial^2 g}{\partial y_s \partial t_s} & \frac{\partial^2 g}{\partial t_s \partial t_s} & \frac{\partial^2 g}{\partial v \partial t_s} \\ \frac{\partial^2 g}{\partial x_s \partial v} & \frac{\partial^2 g}{\partial y_s \partial v} & \frac{\partial^2 g}{\partial t_s \partial v} & \frac{\partial^2 g}{\partial v \partial v} \end{bmatrix} . \tag{36}$$

or, expanded:

$$\frac{\partial^2 g}{\partial \mathbf{m}^2} = \begin{bmatrix} d^{-3}\,(y_r - y_s)^2 / V & -d^{-3}\,(x_r - x_s)\,(y_r - y_s) / V & 0 & d^{-1}\,(x_r - x_s) / V \\ -d^{-3}\,(x_r - x_s)\,(y_r - y_s) / V & d^{-3}\,(x_r - x_s)^2 / V & 0 & d^{-1}\,(y_r - y_s) / V \\ 0 & 0 & 0 & 0 \\ d^{-1}\,(x_r - x_s) / V & d^{-1}\,(y_r - y_s) / V & 0 & d/V \end{bmatrix} . \tag{37}$$

where the notation has been made simpler by using

$$\begin{aligned} d^2 &= (x_r - x_s)^2 + (y_r - y_s)^2 \\ V &= V_0 \exp(v). \end{aligned} \tag{38}$$

Note that, for this example, this matrix is symmetric.

A.3 Second derivatives of $S(\mathbf{m})$

The second derivatives of the misfit function are collected within the Hessian matrix. These derivatives involve second derivatives of the forward operator $g(\mathbf{m})$. The entries of the Hessian of the misfit function, $\hat{\mathbf{H}}$, are defined by (*Tarantola*, 2005, Eq. 6.256, 6.299)

$$\hat{H}_{\alpha\beta}(\mathbf{m}) = \frac{\partial \hat{\gamma}_\alpha}{\partial m^\beta}(\mathbf{m}) = \frac{\partial^2 S}{\partial m^\alpha \partial m^\beta}(\mathbf{m}) \; . \tag{39}$$

In practice, the second-derivative terms are dropped in the approximation for the Hessian, e.g., *Tarantola* (2005, Eq. 6.257). This approximation distinguishes the *Gauss–Newton method* from the *quasi-Newton method*.

References

Hull, D. G., and B. D. Tapley (1977), Square-root variable-metric methods for minimization, *J. Optim. Th. App.*, *21*(3), 251–259.

Liu, Q., and D. Peter (2019), Square-root variable metric based elastic full-waveform inversion—Part 2: uncertainty estimation, *Geophys. J. Int.*, *218*(2), 1100–1120, doi:10.1093/gji/ggz137.

Liu, Q., D. Peter, and C. Tape (2019), Square-root variable metric based elastic full-waveform inversion – Part 1: theory and validation, *Geophys. J. Int.*, *281*(2), 1121–1135, doi: 10.1093/gji/ggz188.

Luo, Y. (2012), Seismic Imaging and Inversion Based on Spectral-Element and Adjoint Methods, Ph.D. thesis, Princeton University, USA.

Tape, C., Q. Liu, and J. Tromp (2007), Finite-frequency tomography using adjoint methods—Methodology and examples using membrane surface waves, *Geophys. J. Int.*, *168*, 1105–1129.

Tarantola, A. (2005), *Inverse Problem Theory and Methods for Model Parameter Estimation*, SIAM, Philadelphia, Penn., USA.

Tarantola, A. (2009), *Mapping of Probabilities: Theory for the Interpretation of Uncertain Physical Measurements*, incomplete manuscript available on-line at `http://www.ipgp.fr/~tarantola/`.

Tarantola, A., and B. Valette (1982), Generalized nonlinear inverse problems solved using the least squares criterion, *Rev. Geophys. Space. Phys.*, *20*(2), 219–232.

Williamson, W. E. (1975), Square-root variable metric method for function minimization, *AIAA Journal*, *13*(1), 107–109.

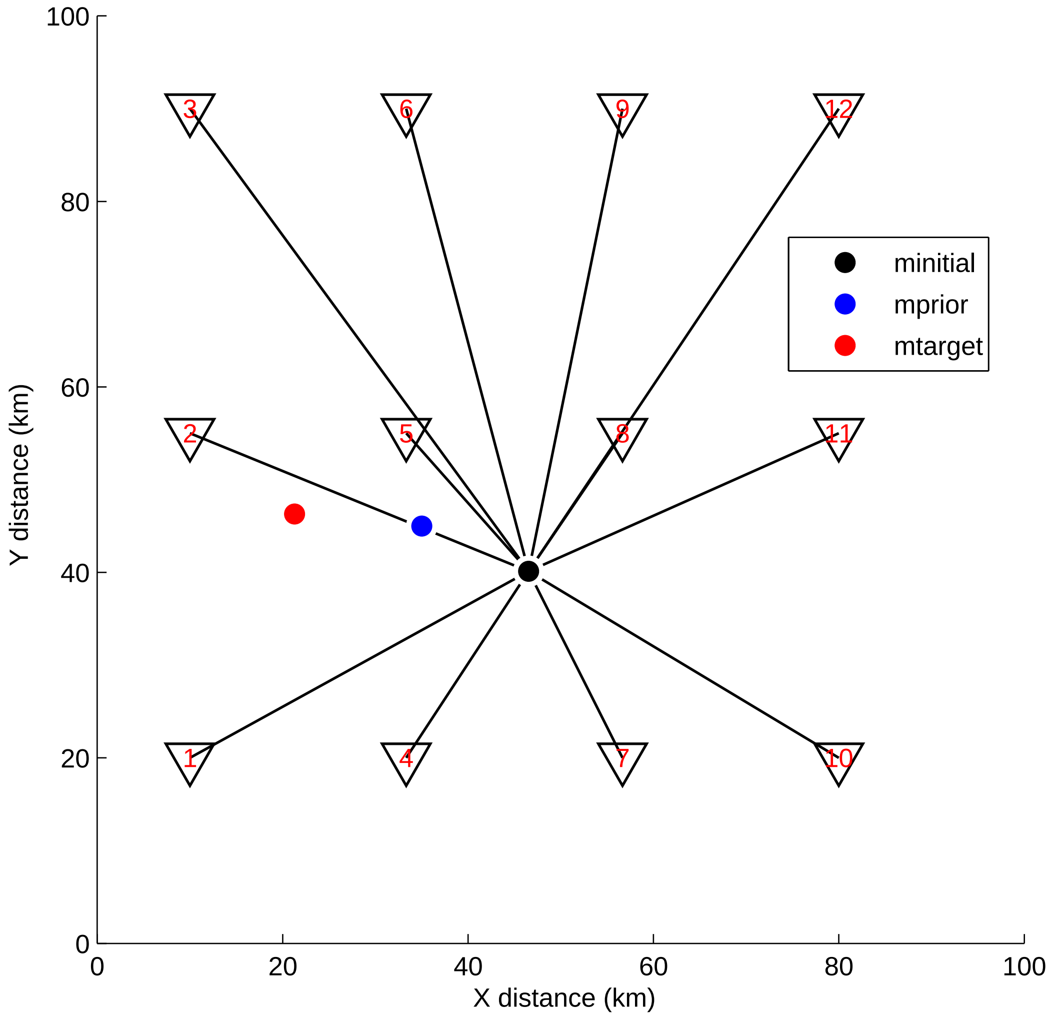

Figure 1: Source–receiver geometry for the epicenter problem. Here, only two of the four model parameters are represented: the epicenter at (x_s, y_s). There are 12 receivers denoted by inverted triangles. See Section 6.1.

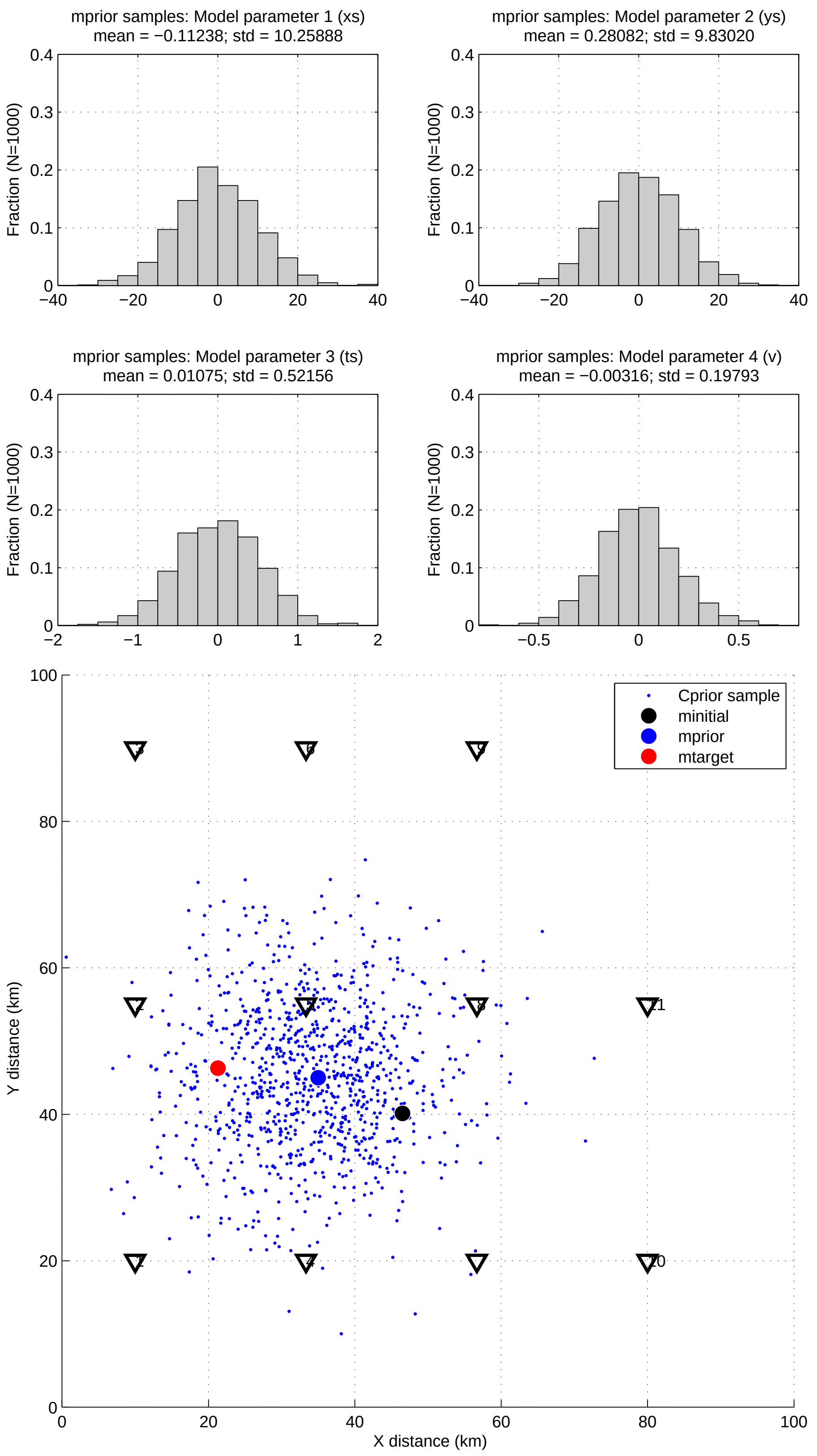

Figure 2: [Example 1 (Section 6.1)] 1000 samples, each a vector having 4 entries, of the prior model covariance matrix (Eq. 4). (Top) Distributions for each model parameter. (Bottom) Physical representation of the two epicenter parameters (x_s, y_s) for all 1000 samples. Also shown is the initial model, $\mathbf{m}_{00}$, and the target model, $\mathbf{m}_{\text{target}}$, for a single inversion run.

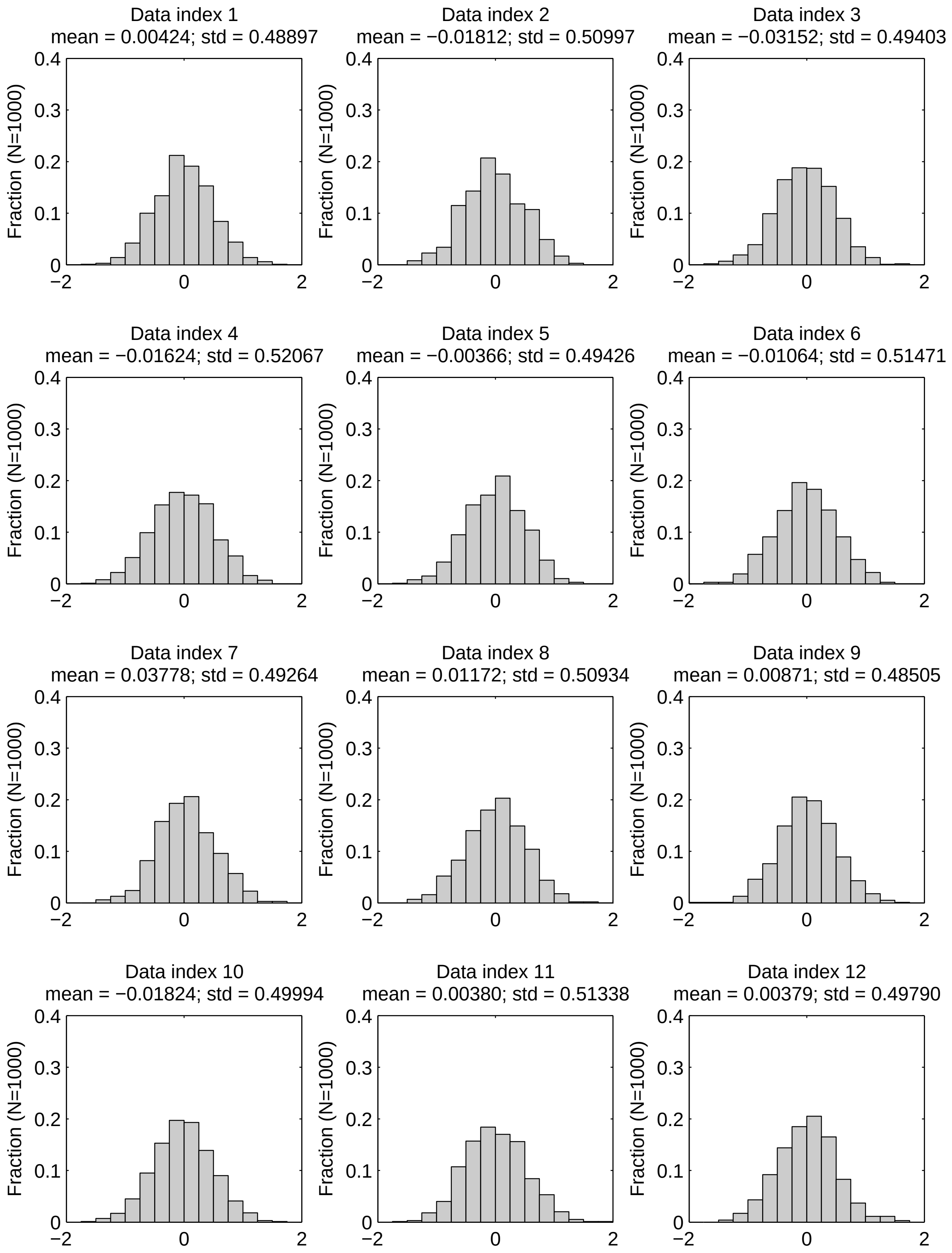

Figure 3: [Example 1 (Section 6.1)] 1000 samples, each a vector having 12 entries, of the data covariance matrix (Eq. 5), plotted showing the distributions for each data index.

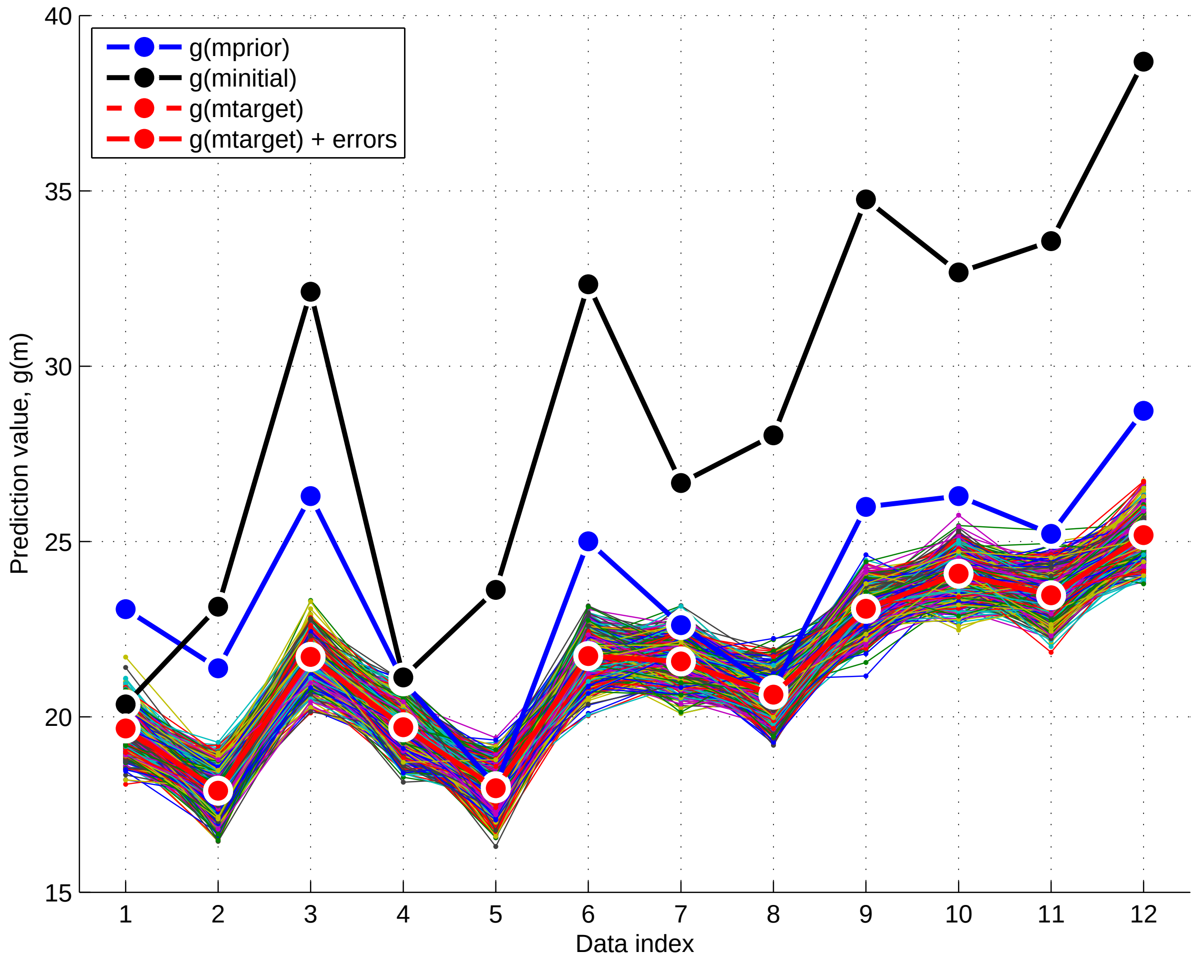

Figure 4: [Example 1 (Section 6.1)] Data vectors. We pick one of 1000 models (Figure 2) to be the target model, $\mathbf{m}_{\text{target}}$, from which we compute the target data, shown as a dashed red line. The "real" data, shown as a solid red line, are computed by adding a randomly selected sample of the data covariance matrix to the target data. The data for the prior model is shown in black.

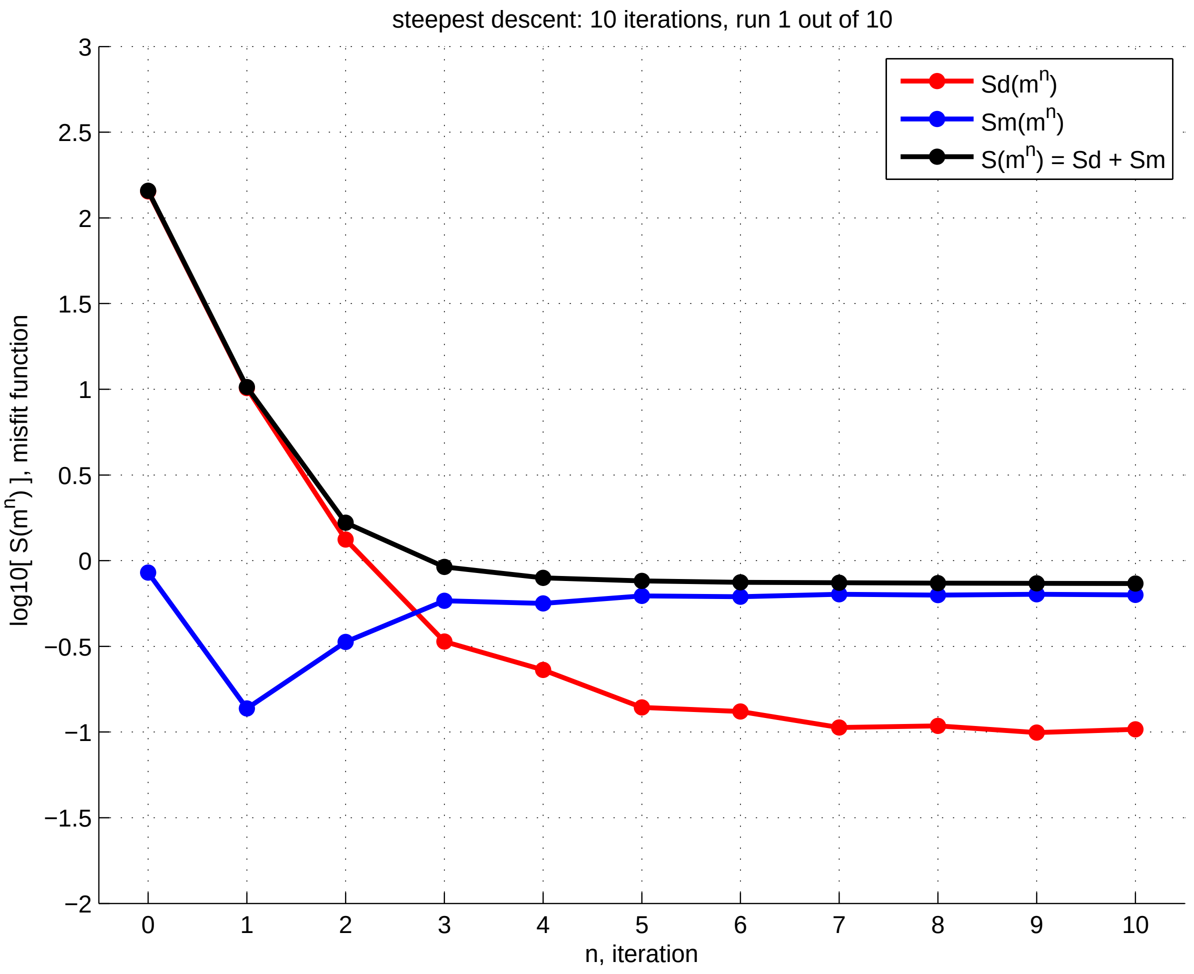

Figure 5: [Example 1 (Section 6.1)] Convergence curve using the steepest descent algorithm.

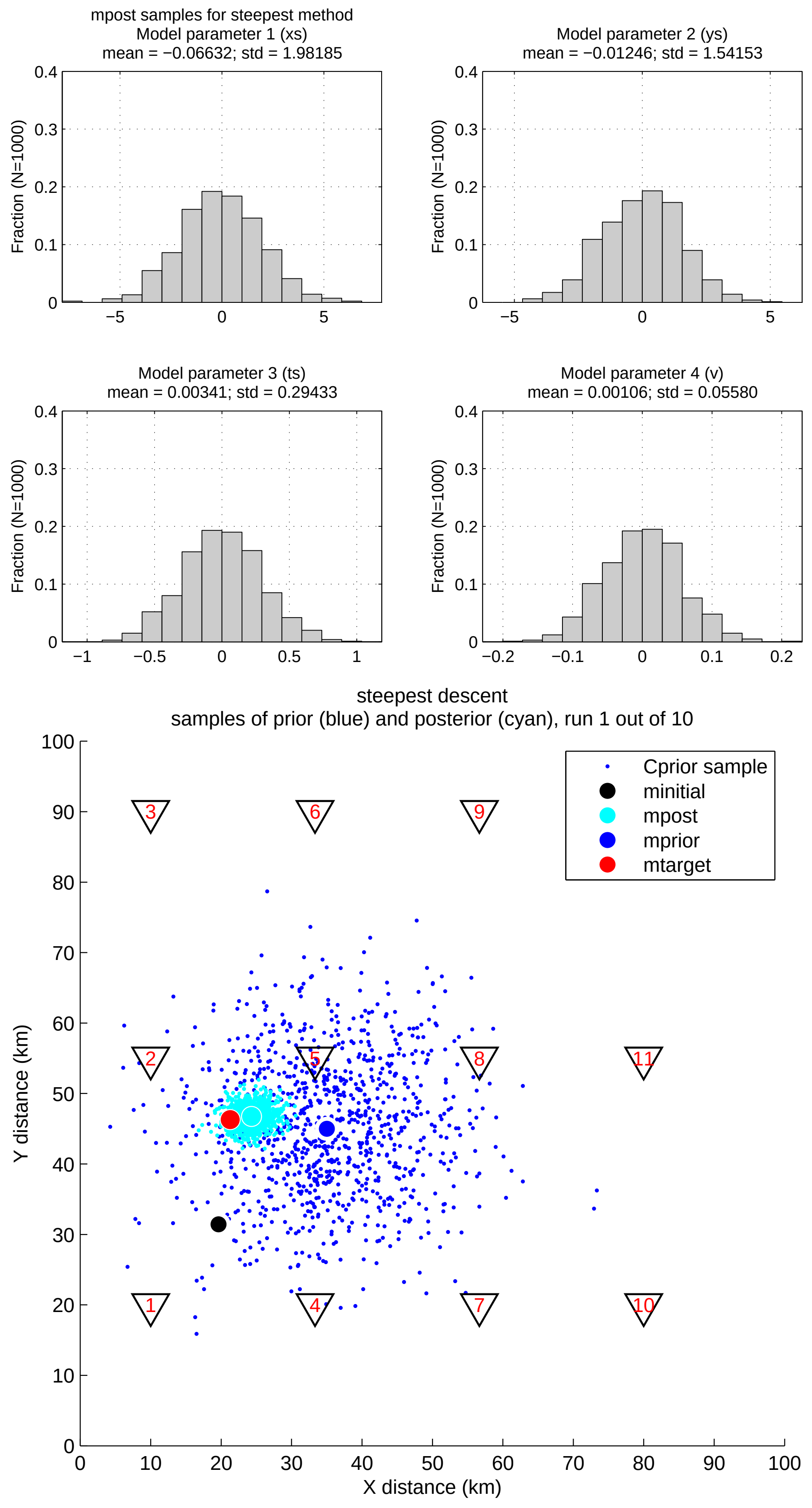

Figure 6: [Example 1 (Section 6.1)] 1000 samples, each a vector having 4 entries, of the posterior model covariance matrix. (Top) Distributions for each model parameter. Note the ranges on the x-axes in comparison with those in Figure 2. (Bottom) Posterior samples (cyan) superimposed on the prior samples (blue), and also showing the initial model, the posterior (final) model ($\mathbf{m}_{10}$), the prior model ($\mathbf{m}_{\rm prior}$), and the target model ($\mathbf{m}_{\rm target}$). The samples of $\mathbf{C}_{\rm post}$ are generated via Equation (2), i.e., they are not produced from the steepest descent algorithm.

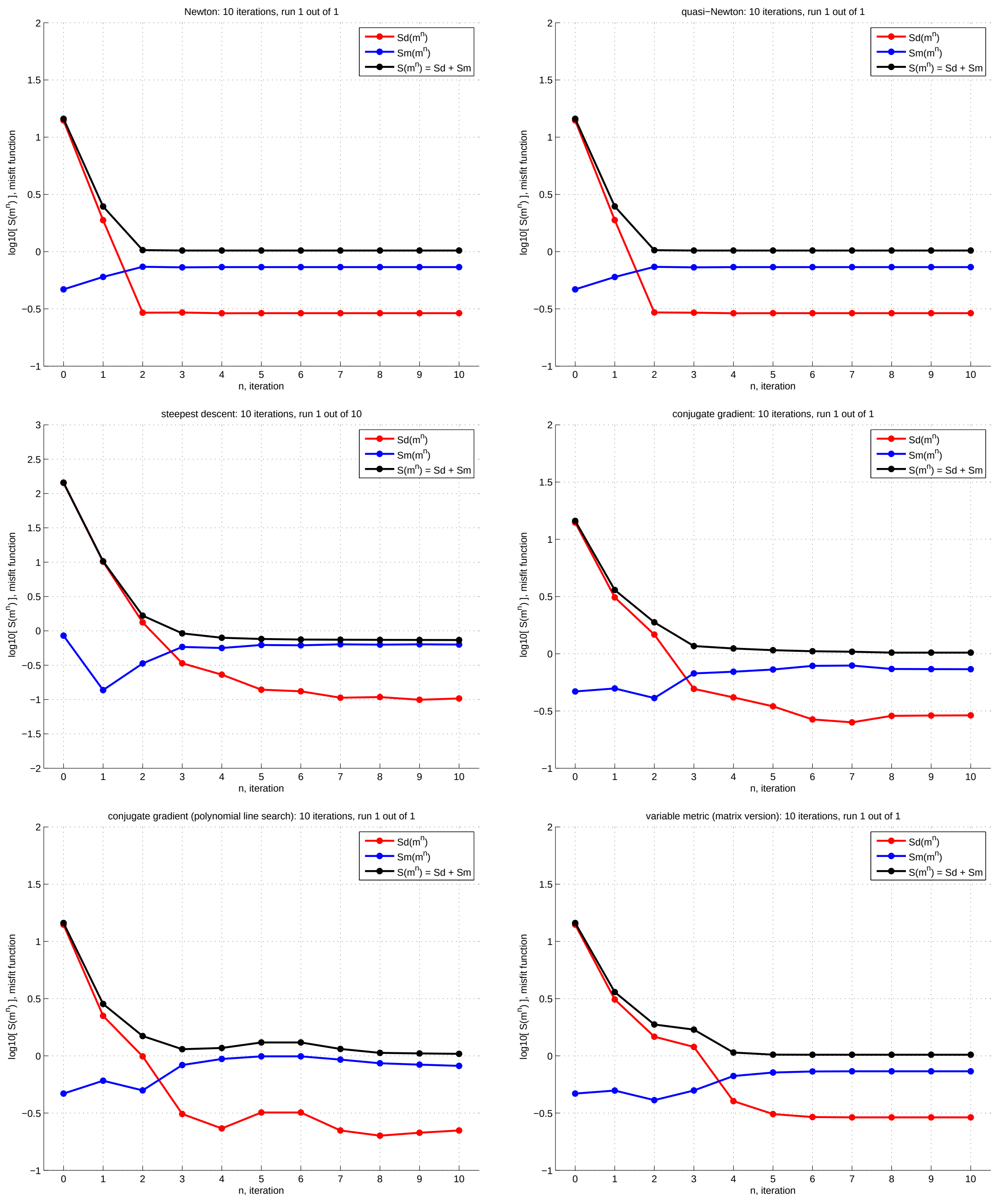

Figure 7: [Example 2a (Section 6.2)] Six representations of posterior distributions of the epicenter parameters (x_s, y_s) for the methods listed in Figure 8; the steepest descent distribution is shown in Figure 6. Note that all methods produce comparable results for $\mathbf{m}_{\text{post}}$. The samples of $\mathbf{C}_{\text{post}}$ are generated via Equation (2).

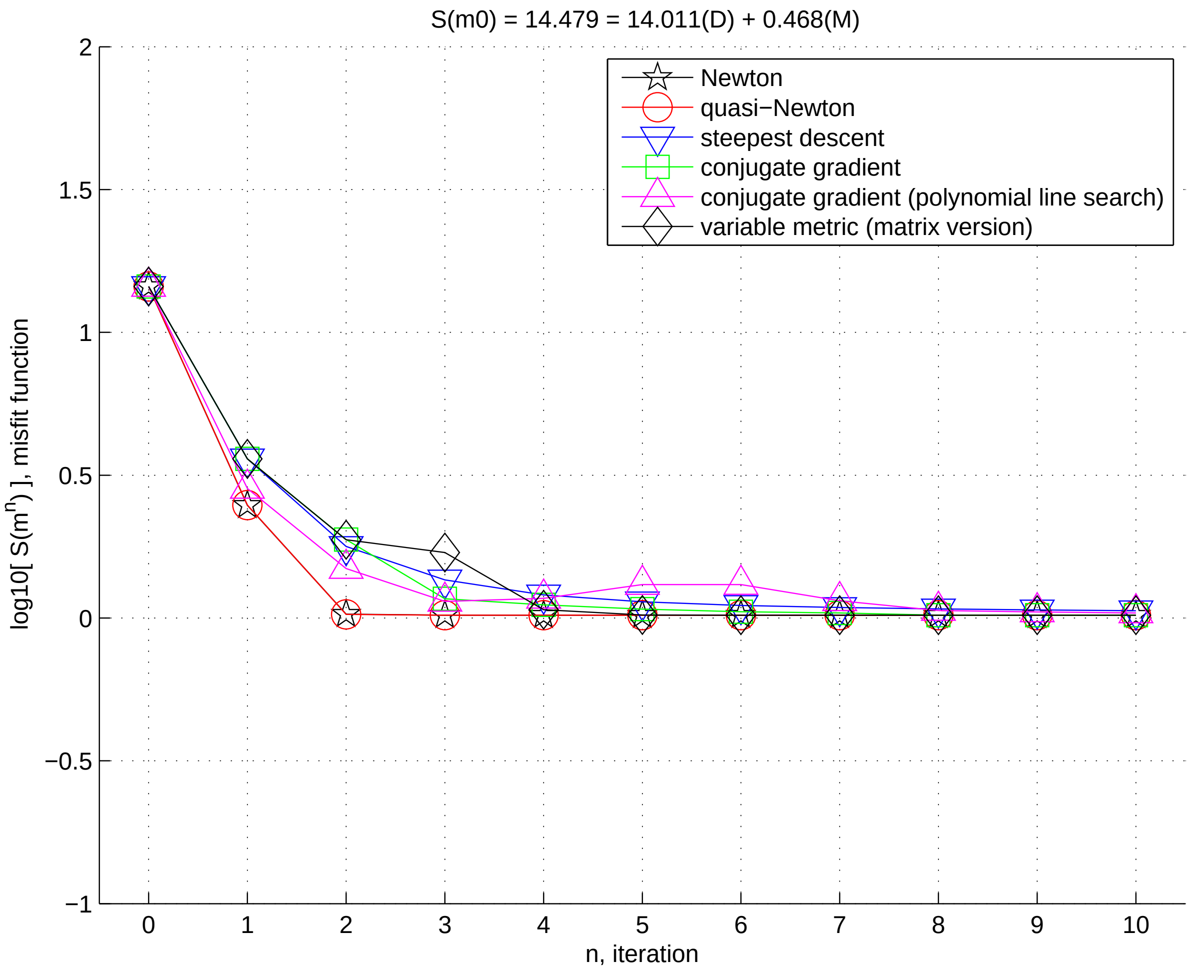

Figure 8: [Example 2a] Convergence curves for six different algorithms for a single run; see Figure 7.

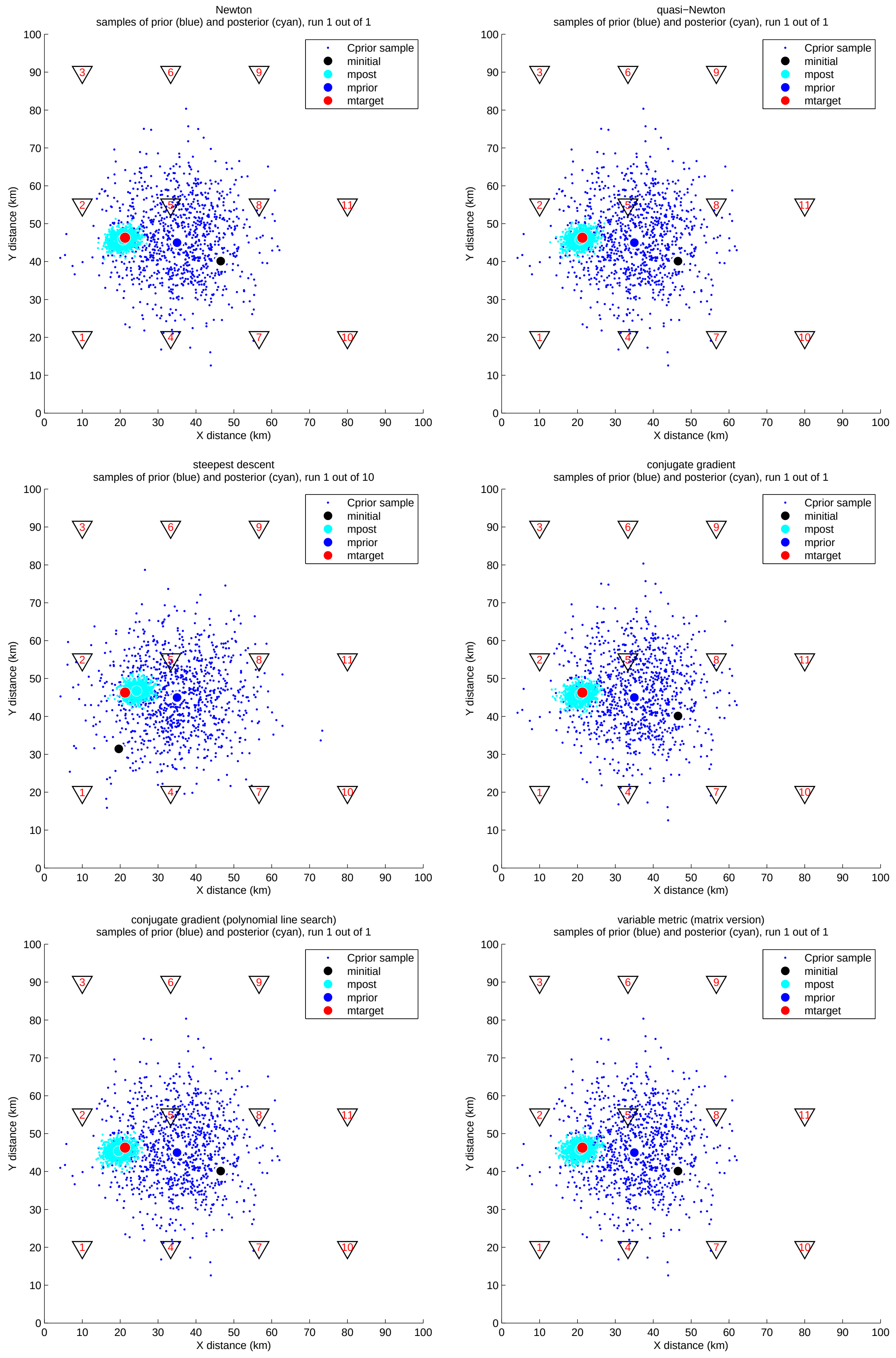

Figure 9: [Example 2a (Section 6.2)] Six representations of posterior distributions of the epicenter parameters (x_s, y_s) for the methods listed in Figure 8; the steepest descent distribution is shown in Figure 6. Note that all methods produce comparable results for $\mathbf{m}_{\text{post}}$. The samples of $\mathbf{C}_{\text{post}}$ are generated via Equation (2).

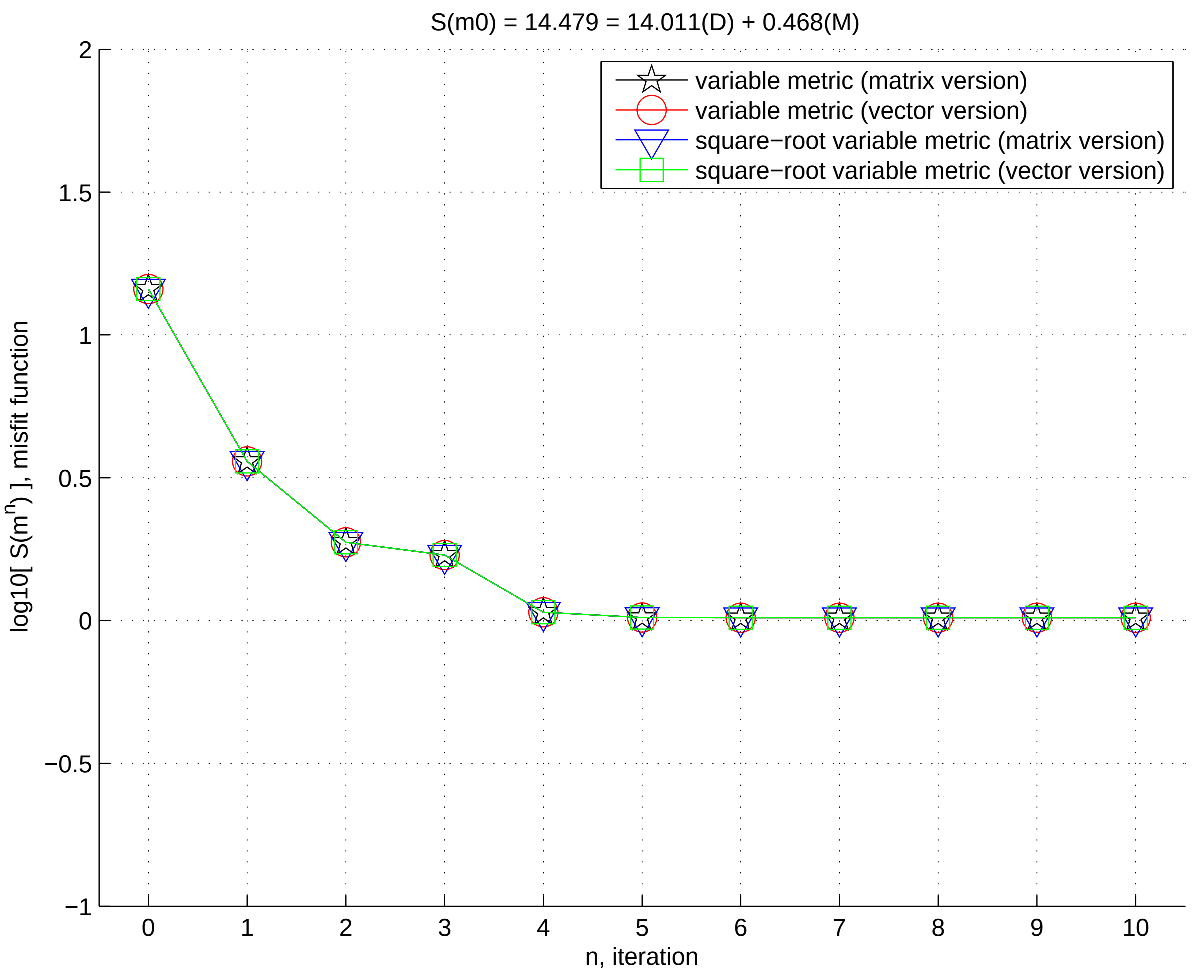

Figure 10: [Example 2b (Section 6.2)] Convergence curves for four variable metric algorithms.

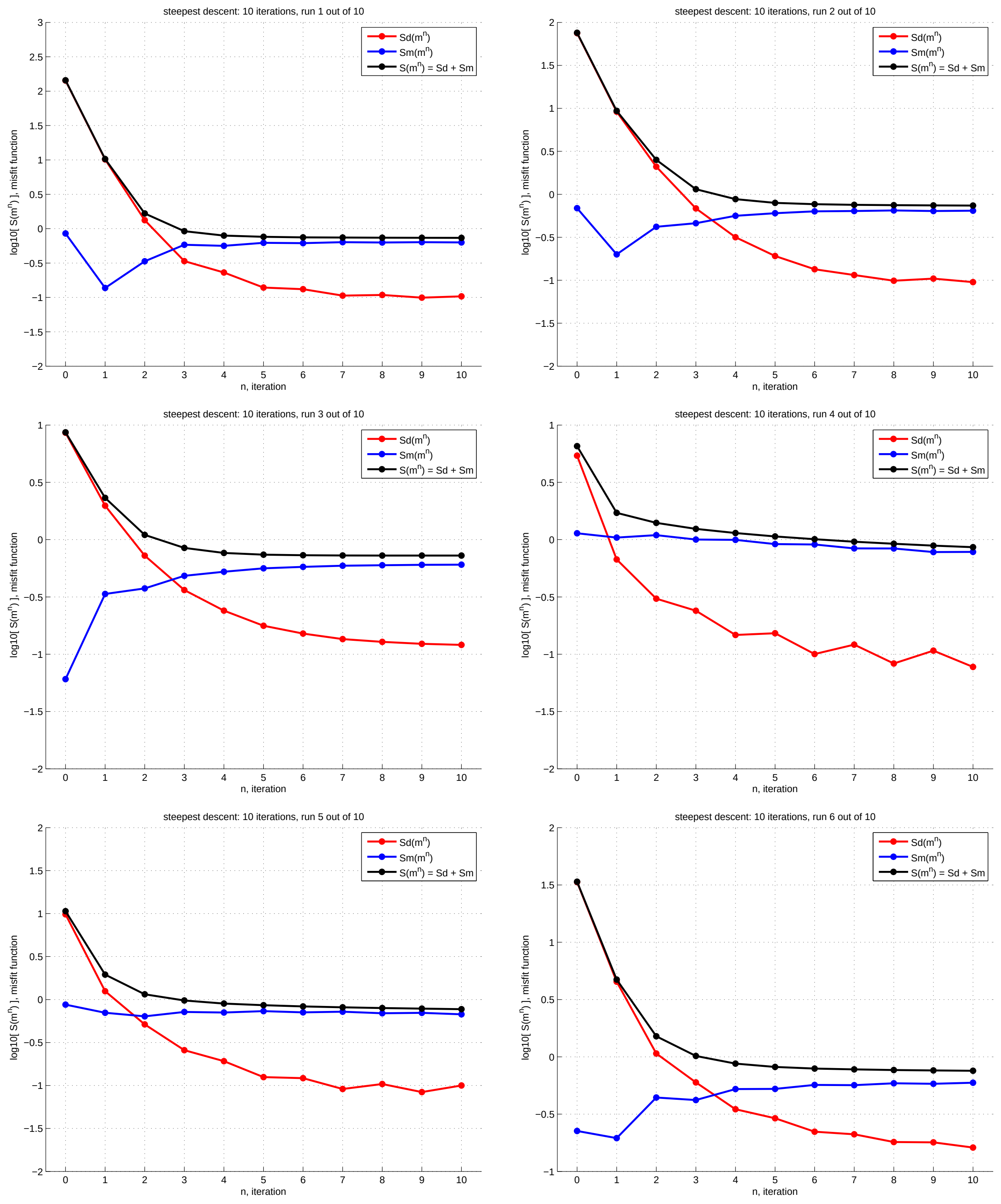

Figure 11: [Example 3a (Section 6.4.)] Convergence curves for steepest descent algorithm for 6 of 10 different initial models using the same fixed target model but with different errors added for each run.

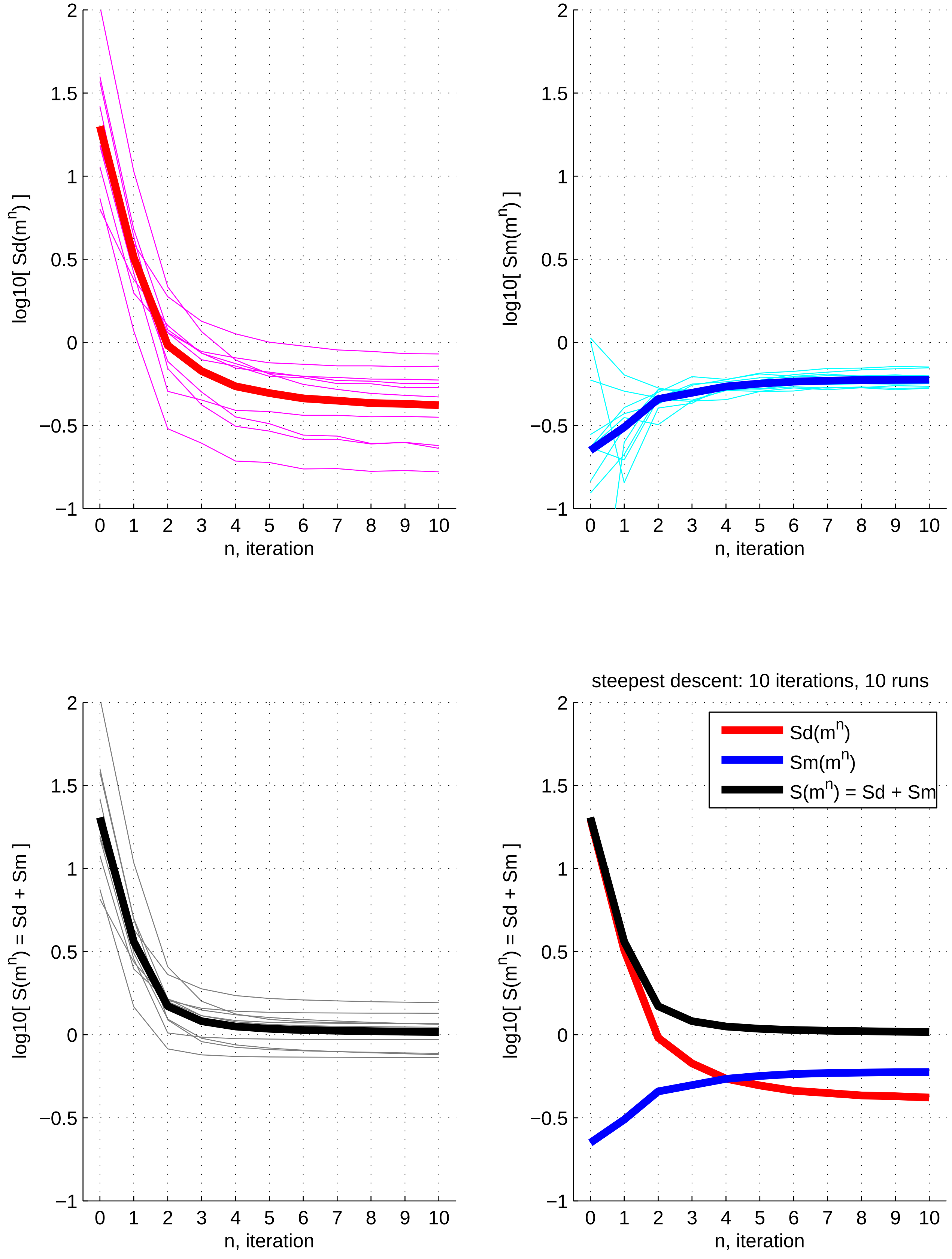

Figure 12: [Example 3a (Section 6.4)] Similar to Figure 11, but collecting the 10 different convergence curves by data misfit (red), prior model misfit (blue), and total misfit (black).

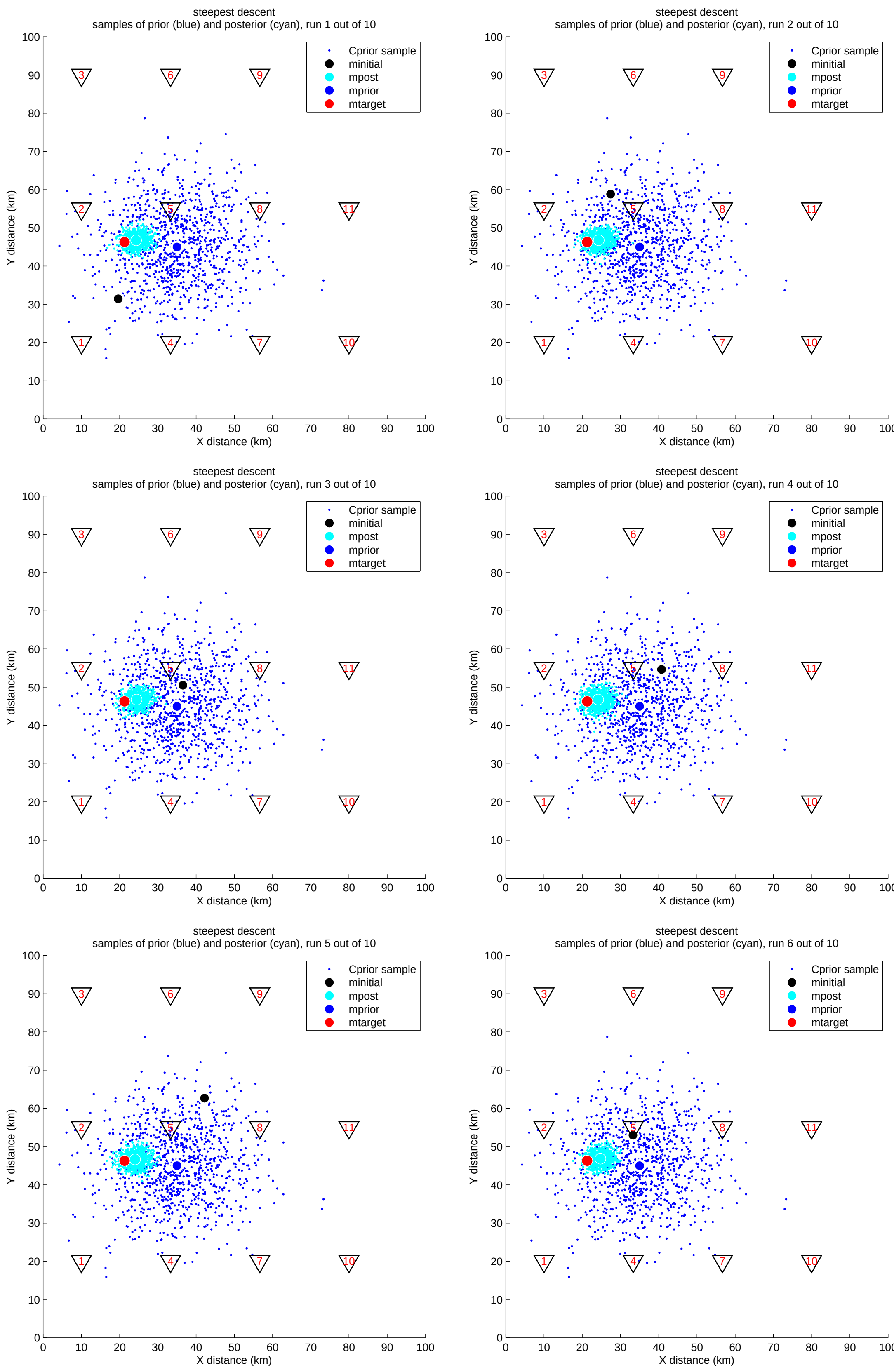

Figure 13: [Example 3a (Section 6.4)] Posterior distributions for 6 of the 10 runs in Example 3 (Section 6.4). Each run starts with a different initial model (black circle) and has a different posterior distribution (cyan dots). The samples of $\mathbf{C}_{\text{post}}$ are generated via Equation (2), i.e., they are not produced from the steepest descent algorithm.

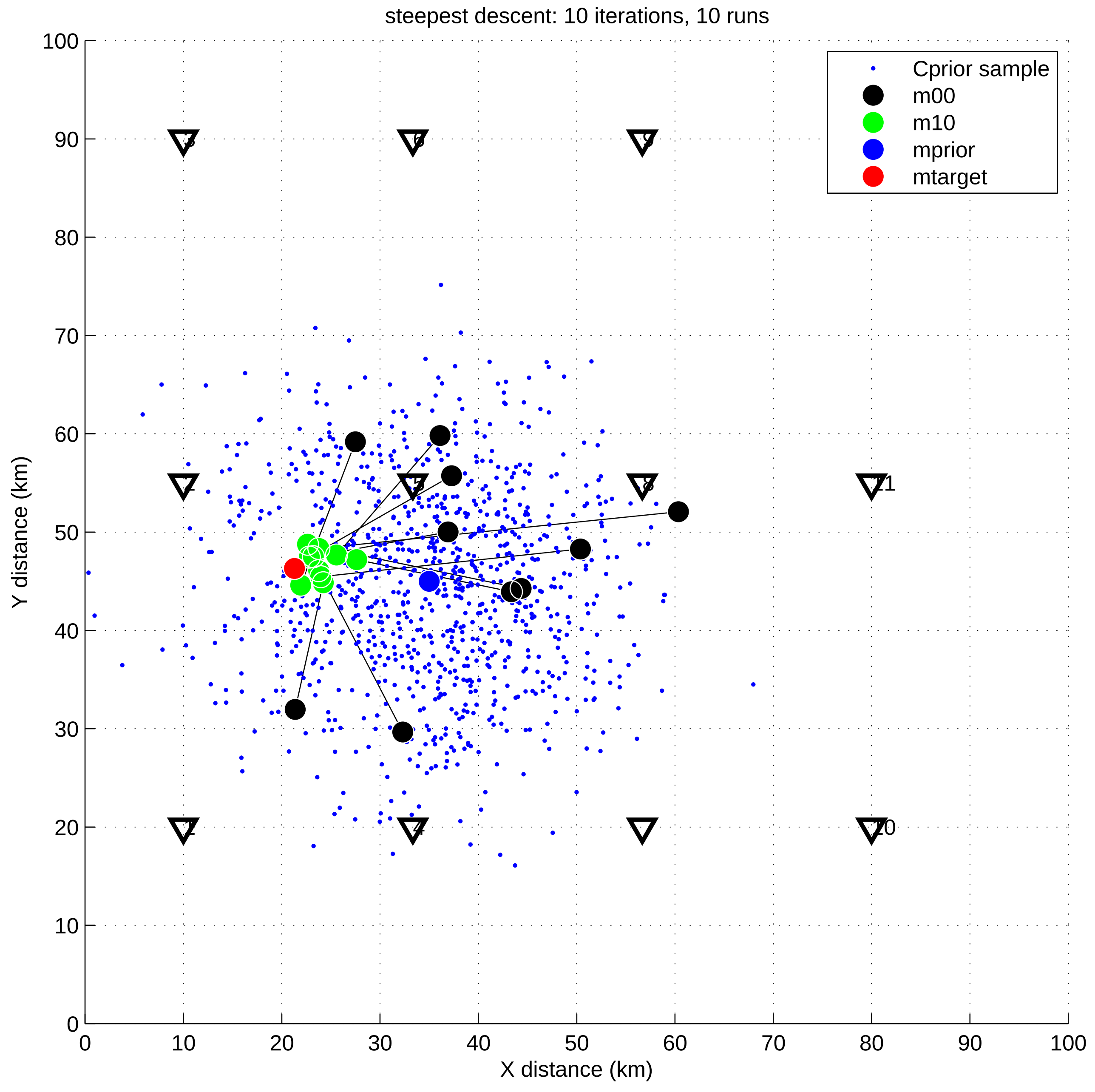

Figure 14: [Example 3a (Section 6.4)] Posterior models for all 10 runs, each with a different initial model and a different set of errors add to data generated from the target model. The segments connect the initial models with the posterior (“final”) models. The posterior distributions for each run are not shown here (see Figure 13).

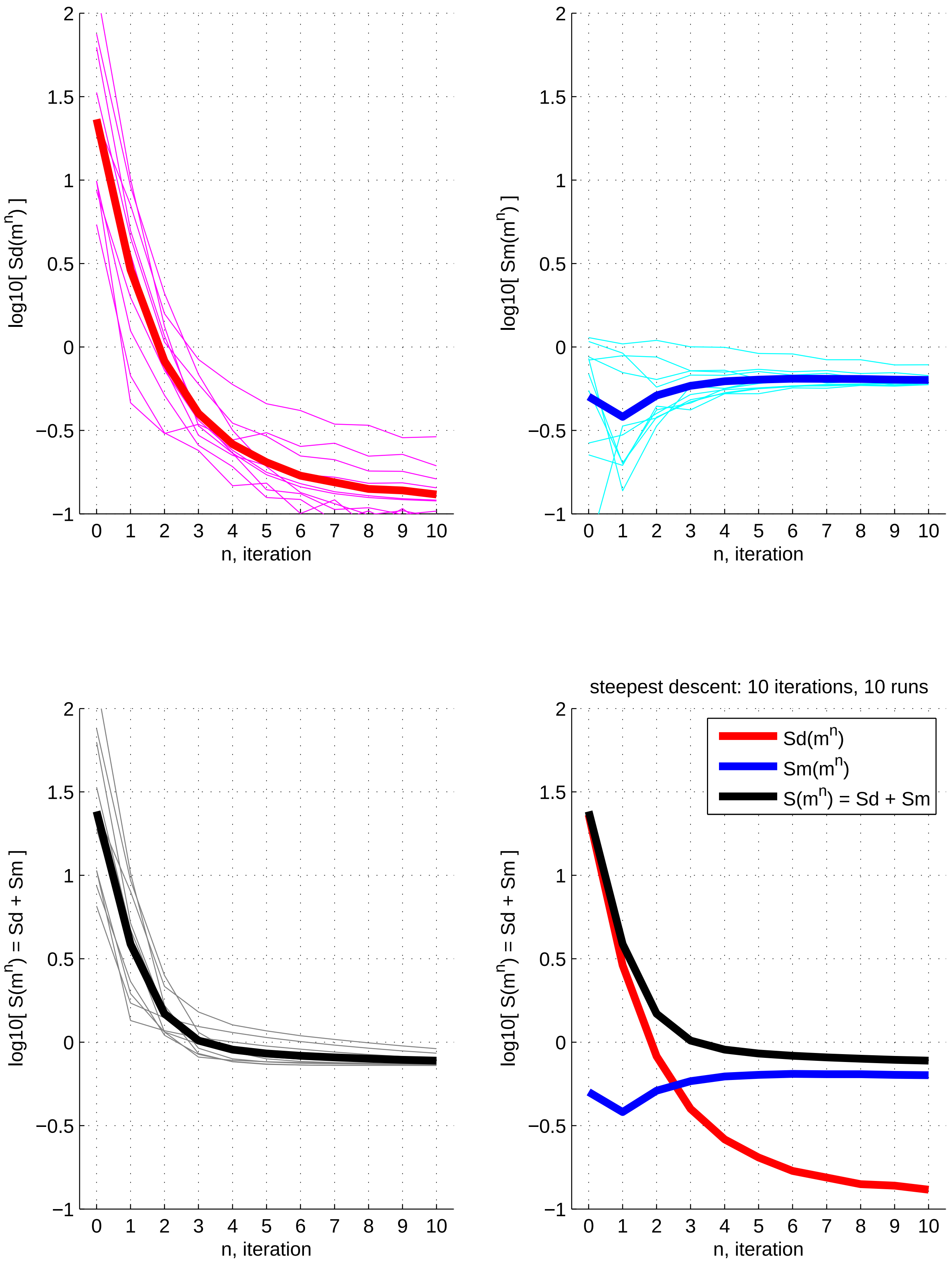

Figure 15: [Example 3b (Section 6.5)] Same as Figure 12 but with no target errors. The 10 initial models (and target errors added) are randomly selected and are different from the 10 runs in Figure 12. Note that the convergence curves $S_{\rm D}(\mathbf{m})$ reach lower values than they do in Figure 12.

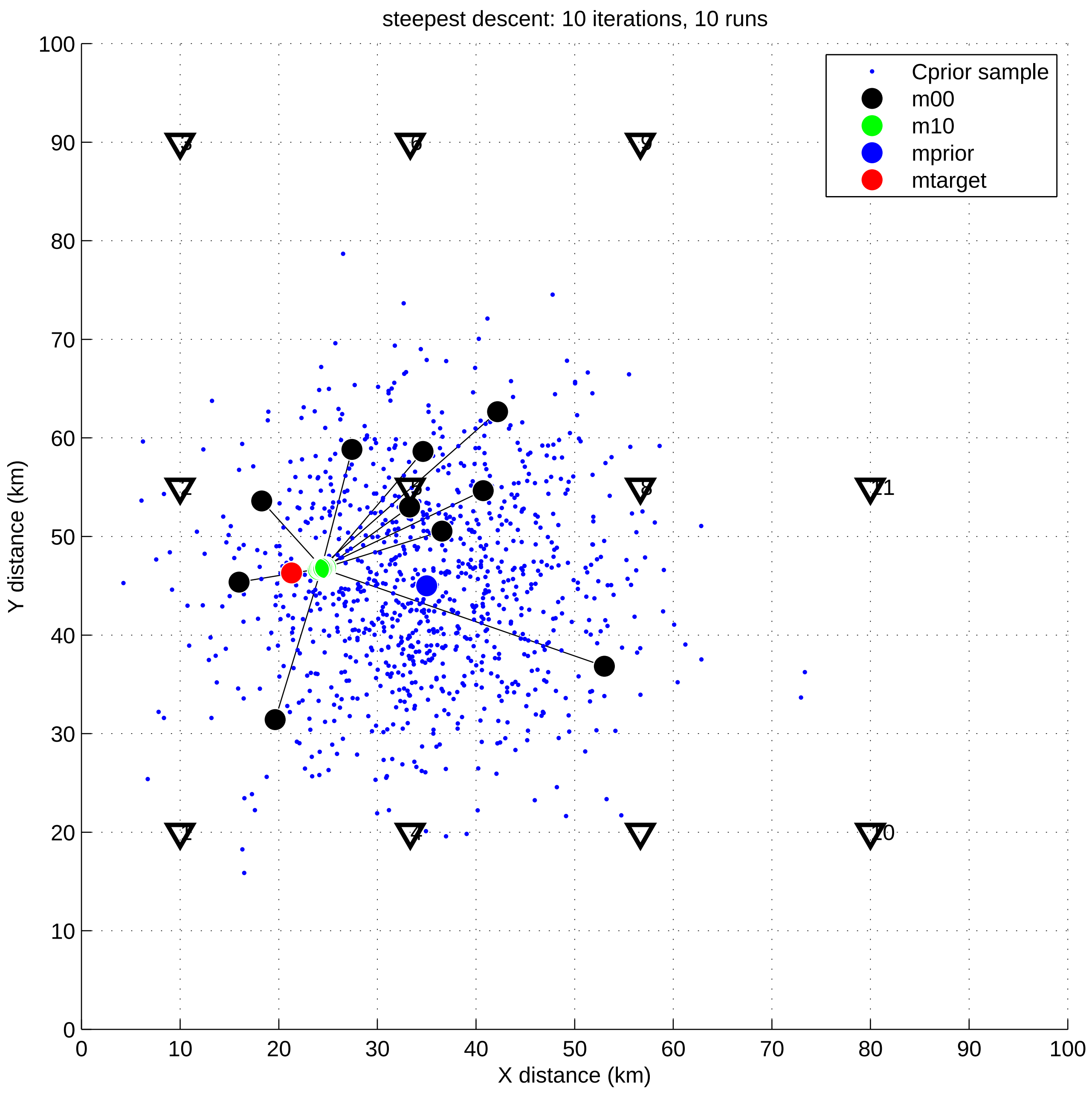

Figure 16: [Example 3b (Section 6.5)] Same as Figure 14 but with no target errors. Because the target errors are the same (zero) for each run, the posterior (final) models are more tightly clustered (than in Figure 14).

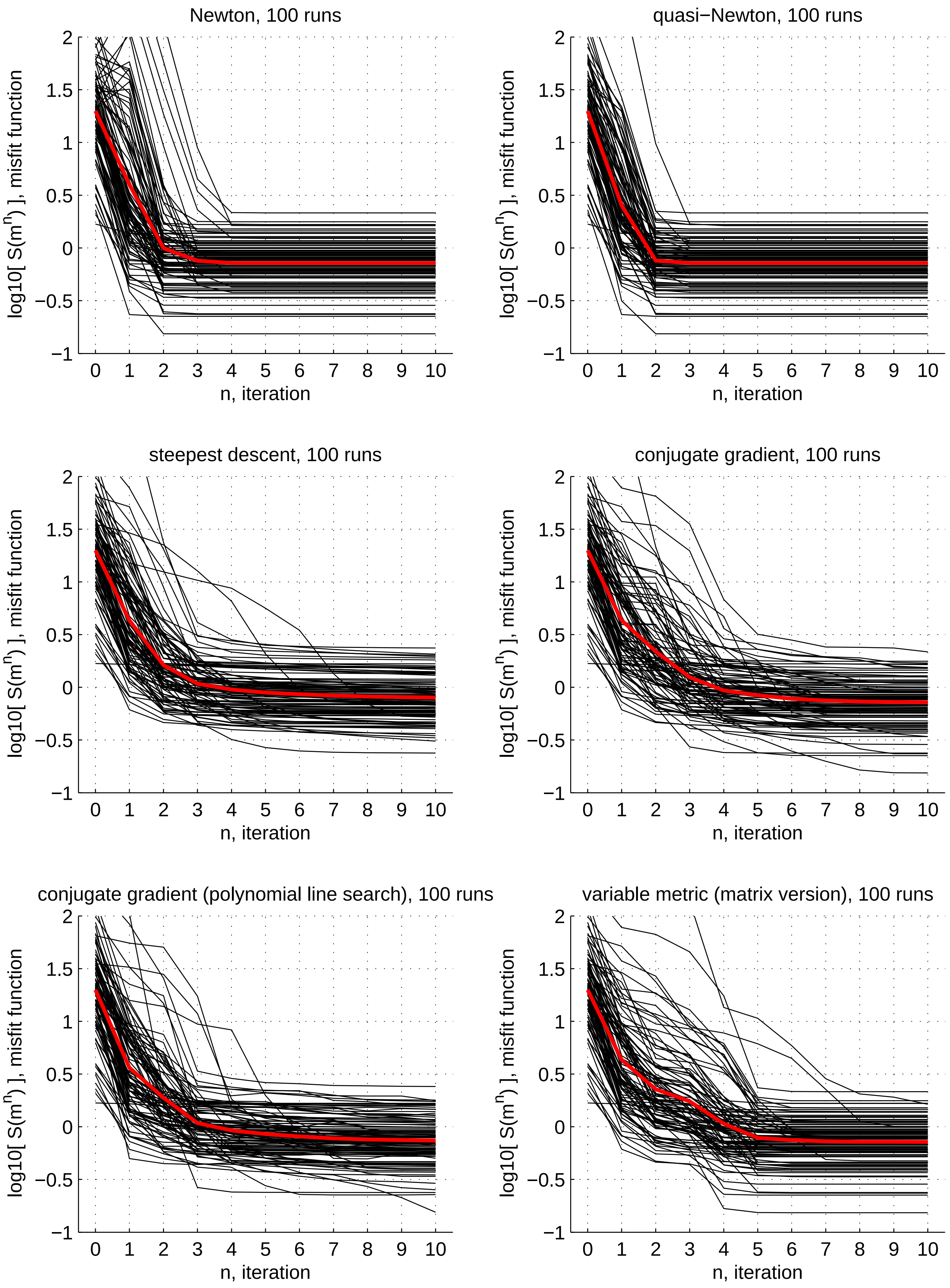

Figure 17: [Example 4 (Section 6.6)] Convergence for six optimization methods for 100 runs. Each run is characterized by a randomly selected initial model, target model, and data errors. The red curve is the mean of all 100 curves in each plot. See Figure 18.

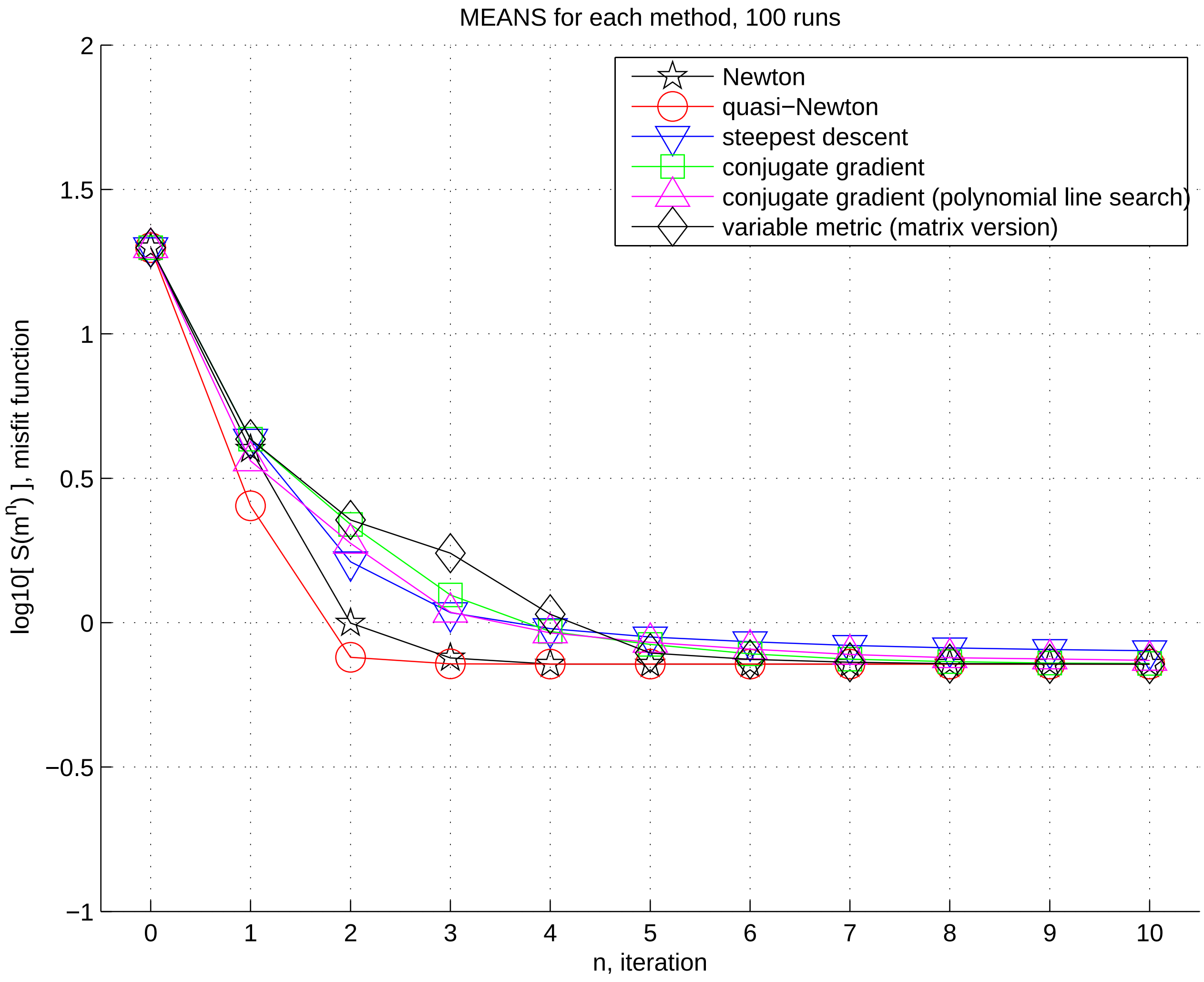

Figure 18: [Example 4 (Section 6.6)] Superposition of the six mean curves in Figure 17.

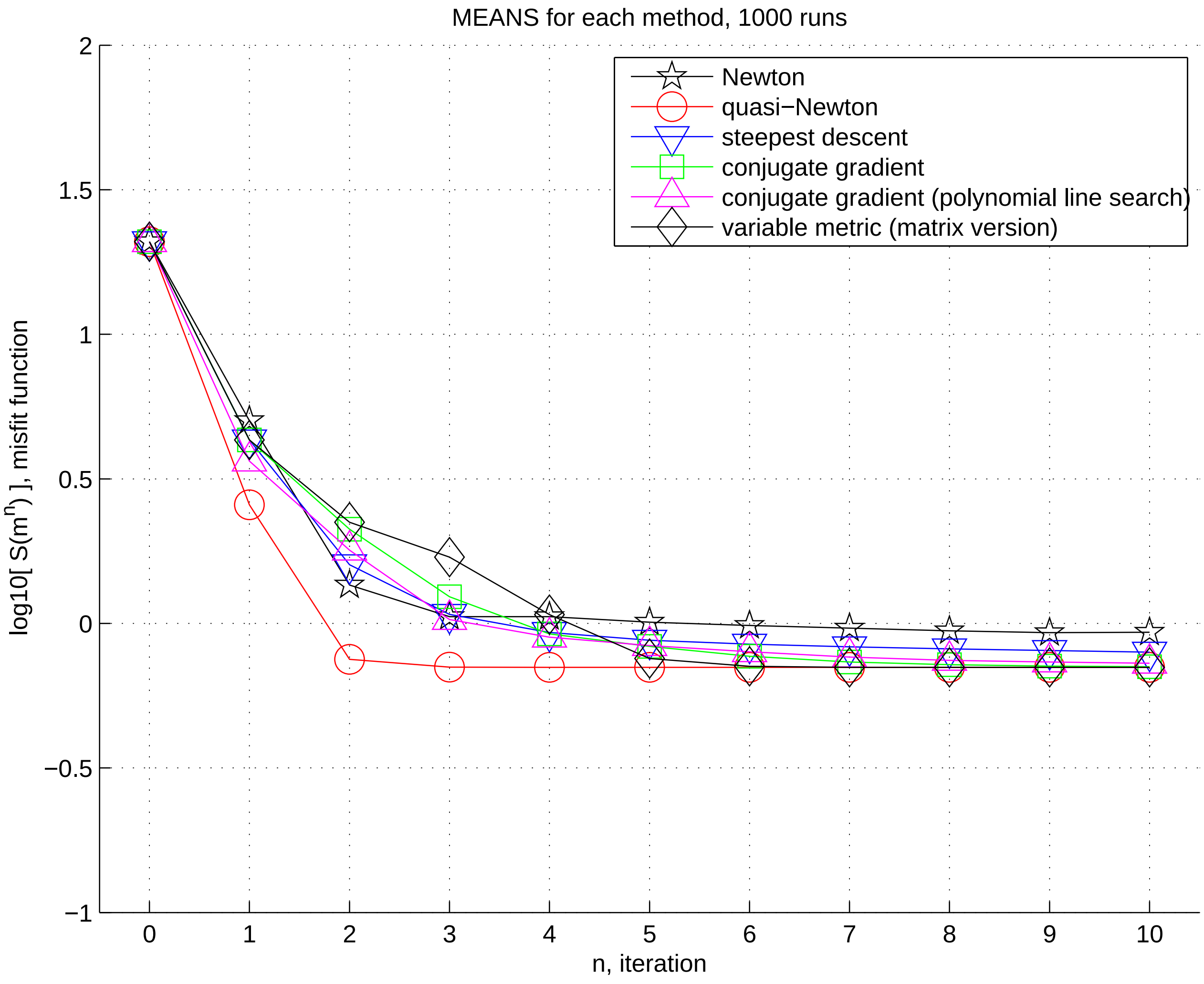

Figure 19: [Example 4 (Section 6.6)] Same as Figure 18 but with 1000 different runs instead of 100.