Optimisation for the estimation of the expansion in spherical harmonics of the antenna patterns

Sébastien Bourguignon

March 14, 2007

Introduction

We consider the expansion in spherical harmonics (ESH) of the k^{th} antenna pattern (AP):

$$F_k = \widetilde{F}_k + \epsilon_k, \text{ with } \widetilde{F}_k = \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} C_{\ell m}^k Y_{\ell m}$$

where $C_{\ell m}^k \in \mathbf{C}$ are the coefficients of the expansion, $\mathbf{Y}_{\ell m}$ are the spherical harmonics and ϵ_k is the residual after decomposition. There is a total of $D = (\ell_{\max}+1)^2$ coefficients.

In practice, all quantities are computed using the hexagonal map $(x_m, y_m)_{m=1\ldots M}$, made of $M = 34087$ points (inside the unitary circle). In the following, we will use the matrix-vector formulation:

$$\boldsymbol{F}_k = \widetilde{\boldsymbol{F}}_k + \boldsymbol{\epsilon}_k, \text{ with } \widetilde{\boldsymbol{F}}_{\mathbf{k}} = \mathbf{Y}\boldsymbol{C}_k$$

where $\boldsymbol{F}_k$ (resp. $\widetilde{\boldsymbol{F}}_k$) collects the values of the corresponding AP in a $M \times 1$ column vector, $\mathbf{Y}$ is the $M \times D$ matrix containing the spherical harmonics tabulated on the (x, y) map and $\boldsymbol{C}_k$ is a $D \times 1$ column vector containing the coefficients of the decomposition. There are $K = 69$ antennas. Noting $\mathbf{F}$ (resp. $\widetilde{\mathbf{F}}$) the $M \times K$ matrix whose columns are formed by the $\boldsymbol{F}_k$ (resp. $\widetilde{\boldsymbol{F}}_k$), and noting $\mathbf{C}$ the $D \times K$ matrix whose columns are formed by the $\boldsymbol{C}_k$, we have:

$$\widetilde{\mathbf{F}} = \mathbf{YC}. \tag{1}$$

Given an observed scene $\boldsymbol{T}$, the measurements of the N visibilities $V_{b,\, b=1\ldots N}$ are obtained by $\boldsymbol{V} = \mathbf{G}(\mathbf{F})\boldsymbol{T}$, where $\mathbf{G}(\mathbf{F})$ is the $N \times M$ matrix with elements:

$$G_{b,m}(\mathbf{F}) = \frac{S_{xy}}{\sqrt{1 - x_m^2 - y_m^2}} F_{\ell(b)}(x_m, y_m)^* F_{k(b)}(x_m, y_m) e^{-j2\pi(u(b)x_m + v(b)y_m)} \tag{2}$$

where S_{xy} is a constant, $(k(b), l(b))$ indicate the couple of antennas involved in the b^{th} element of $\boldsymbol{V}$ and $(u(b), v(b))$ are[1]. We have $N = 2349 = 69 * 68/2 + 3$[2].

[1] a completer

[2] Eric: pourquoi le +3??

In practice: First, the spherical harmonics were tabulated on the (x, y) map. The corresponding script `tabYlm_xietaNEW.m` *and results* `Ylmtab_xietaNEW.mat` *are stored in SMOS/Spherical.*

In fine, we want to be able to estimate the best expansions in spherical harmonics for all antenna patterns from the measurements of the visibilities $\boldsymbol{V}_{\rm exp}$ corresponding to a *perfectly known* scene $\boldsymbol{T}$. That it, we want to minimize:

$$J(\mathbf{C}) = \left\| \boldsymbol{V}_{\rm exp} - \widetilde{\boldsymbol{V}} \right\|^2 = \left\| \boldsymbol{V}_{\rm exp} - \mathbf{G}(\widetilde{\mathbf{F}})\boldsymbol{T} \right\|^2$$

where $\mathbf{G}(\widetilde{\mathbf{F}})$ is obtained from (2) with $\widetilde{\mathbf{F}} = \mathbf{YC}$.

Computation of J

From (2), it comes that:

$$\begin{aligned}
\widetilde{V}_b &= \sum_m G_{b,m}(\widetilde{\mathbf{F}}) T(x_m, y_m) \\
&= \sum_m \frac{S_{xy}}{\sqrt{1 - x_m^2 - y_m^2}} \widetilde{F}_{\ell(b)}(x_m, y_m)^* \widetilde{F}_{k(b)}(x_m, y_m) e^{-j2\pi(u(b)x_m + v(b)y_m)} T(x_m, y_m) \\
&= \widetilde{\mathbf{F}}_{\ell(b)}^\dagger \mathbf{D}_b(\boldsymbol{T}) \widetilde{\mathbf{F}}_{k(b)},
\end{aligned}$$

where $\mathbf{D}_b(\boldsymbol{T})$ is a $M \times M$ diagonal matrix whose $m^{\rm th}$ element is:

$$\frac{S_{xy}}{\sqrt{1 - x_m^2 - y_m^2}} e^{-j2\pi(u(b)x_m + v(b)y_m)} T(x_m, y_m),$$

so that for a given scene $\boldsymbol{T}$ all matrices $\mathbf{D}_b(\boldsymbol{T})$ can be computed once for all. In the following, we omit the dependence in $\boldsymbol{T}$ of matrices $\mathbf{D}_b$.

In practice: Script `CG/matrices/calcDb.m` *computes the matrices $\mathbf{D}_b(\boldsymbol{T})$ corresponding to a given temperature map in a given polarisation. The $\mathbf{D}_b$ are stored in files such as* `SMOS/CG/matrices/Db/Db_Tsky1_polh.mat`*, as a 34087×2349 matrix (recall that the Db are diagonal). The resulting file size is 1.2 Go, so it cannot be loaded with any machine...*

With $\widetilde{\mathbf{F}} = \mathbf{YC}$, we have:

$$\widetilde{V}_b = \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)},$$

where β_b is the $D \times D$ matrix:

$$\beta_b = \mathbf{Y}^\dagger \mathbf{D}_b \mathbf{Y}. \tag{3}$$

For a given scene $\boldsymbol{T}$ and a given number of spherical harmonics, all matrices β_b can be computed once for all using the previously computed $\mathbf{D}_b$, which do not depend on which spherical harmonics are considered.

In practice: Script `CG/matrices/calcDb.m` *also computes matrices β_b for a given scene and a given polarisation, and for given ℓ_{max} and m_{max}. The β_b are stored in files such as* `SMOS/CG/matrices/Beta/Beta_Tsky1_polh_lmax5_mmax5.mat`*. For $\ell_{max} = 5$, $m_{max} = 5$, the file size is 39 Mo.*

Once matrices β_b are computed, the evaluation of criterion J reduces to:

$$J(\mathbf{C}) = \sum_b \left| V_{\exp_b} - \mathbf{C}^\dagger_{\ell(b)} \beta_b \mathbf{C}_{k(b)} \right|^2 . \tag{4}$$

In practice: Function `calc_J.m` *computes the criterion.*

Criterion J is the sum of squares of quadratic terms in the unknown coefficients $C^k_{\ell,m}$, so it writes as a polynomials of order 4 in all $C^k_{\ell,m}$. Given the high dimension of the problem (there are $D \times K$ coefficients, with $D = (\ell_{\max}+1)^2$ the number of harmonics and K the number of antennas), it is probably full of local minima.

A first objective is to develop optimisation algorithms to reach a local minimum of J. If the algorithm is initialised close to the global minimiser, it should be able to retrieve the latter.

1 Optimisation strategies: gradient-based methods

Expression (4) is complicated, but allows to derive an analytical expression for the gradient of J. This allows to tackle the optimisation problem by a wide range of methods.

1.1 Computation of the gradient

We have $J(\mathbf{C}) = \sum_b J_b = \sum_b {A_b}^\dagger A_b$, with $A_b = V_b - \mathbf{C}^\dagger_{\ell(b)} \beta_b \mathbf{C}_{k(b)}$. Let $\boldsymbol{C}_0$ denote the vector of coefficients for a given antenna k_0. We have:

$$\frac{\partial J_b}{\partial \boldsymbol{C}_0^r} = \frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^r} A_b + A_b^\dagger \frac{\partial A_b}{\partial \boldsymbol{C}_0^r} A_b$$
$$\frac{\partial J_b}{\partial \boldsymbol{C}_0^i} = \frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^i} A_b + A_b^\dagger \frac{\partial A_b}{\partial \boldsymbol{C}_0^i} A_b$$

where $\boldsymbol{C}_0^r$ and $\boldsymbol{C}_0^i$ denote the real and imaginary parts of $\boldsymbol{C}_0$, respectively. Thus, $\overrightarrow{\nabla} J(\mathbf{C}) = \sum_b \overrightarrow{\nabla} J_b(\mathbf{C})$, with $\overrightarrow{\nabla} J_b(\mathbf{C}) = \frac{\partial J_b}{\partial \boldsymbol{C}_0^r} + j \frac{\partial J_b}{\partial \boldsymbol{C}_0^i}$. To compute $\overrightarrow{\nabla} J_b(\mathbf{C})$, we separate the indexes b in four sets:

- B_1 collects the indexes b such that $k(b) = \ell(b) = k_0$;
- B_2 collects the indexes b such that $k(b) = k_0$ and $\ell(b) \neq k_0$;
- B_3 collects the indexes b such that $k(b) \neq k_0$ and $\ell(b) = k_0$;
- B_4 collects the remaining indexes, for which A_b does not depend on $\boldsymbol{C}_0$.

For $b \in B_1$, we have $A_b = V_b - \mathbf{C}_{k_0}^\dagger \beta_b \mathbf{C}_{k_0}$ and[3]:

$$\begin{aligned}
\frac{\partial A_b}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_0^{r'}(\beta_b + \beta_b') + j\boldsymbol{C}_0^{i'}(\beta_b - \beta_b') \\
\frac{\partial A_b}{\partial \boldsymbol{C}_0^i} &= -\boldsymbol{C}_0^{i'}(\beta_b + \beta_b') - j\boldsymbol{C}_0^{r'}(\beta_b - \beta_b') \\
\frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_0^{r'}(\beta_b' + \beta_b^*) + j\boldsymbol{C}_0^{i'}(\beta_b.' - \beta_b^*) \\
\frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^i} &= -\boldsymbol{C}_0^{i'}(\beta_b.' + \beta_b^*) - j\boldsymbol{C}_0^{r'}(\beta_b.' - \beta_b^*)
\end{aligned}$$

For $b \in B_2$, we have $A_b = V_b - \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k_0}$ and:

$$\begin{aligned}
\frac{\partial A_b}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_{\ell(b)}^\dagger \beta_b \qquad \frac{\partial A_b}{\partial \boldsymbol{C}_0^i} = -j\boldsymbol{C}_{\ell(b)}^\dagger \beta_b \\
\frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_{\ell(b)}' \beta_b^* \qquad \frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^i} = j\boldsymbol{C}_{\ell(b)}' \beta_b^*
\end{aligned}$$

For $b \in B_3$, we have $A_b = V_b - \mathbf{C}_{k_0}^\dagger \beta_b \mathbf{C}_{k(b)}$ and:

$$\begin{aligned}
\frac{\partial A_b}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_{k(b)}' \beta_b' \qquad \frac{\partial A_b}{\partial \boldsymbol{C}_0^i} = j\boldsymbol{C}_{k(b)}' \beta_b' \\
\frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^r} &= -\boldsymbol{C}_{k(b)}^\dagger \beta_b^\dagger \qquad \frac{\partial A_b^\dagger}{\partial \boldsymbol{C}_0^i} = -j\boldsymbol{C}_{k(b)}^\dagger \beta_b^\dagger
\end{aligned}$$

For $b \in B_4$, we have $\frac{\partial J_b}{\partial \boldsymbol{C}_0} = 0$.

Consequently, the gradient $\overrightarrow{\nabla} J(\mathbf{C})$ can be computed in a loop on $k_0 \in 1 \dots K$ at a very reasonable cost: for every k_0, select the corresponding sets B_1-B_4 and compute the appropriate sum. [4]

In practice: Function `gradJ.m` *computes the gradient.*

1.2 Gradient algorithm

The gradient algorithm, at iteration t, consists in the following steps:

1. compute the gradient $\overrightarrow{\nabla} J(\mathbf{C}^{(t)})$;

[3] Moreover, β_b is hermitian here.

[4] The above expressions were found empirically to simplify a little bit (see the matlab code). I still have to verify analytically why...

2. choose the descent direction $\mathbf{D}^{(t)} = -\overrightarrow{\nabla} J(\mathbf{C}^{(t)})$
3. perform line search: find the optimal value $\alpha_{\min} > 0$ such that $\alpha_{\min} = \arg\min_\alpha J(\mathbf{C}^{(t)} + \alpha\mathbf{D}^{(t)})$
4. update $\mathbf{C}^{(t+1)} = \mathbf{C}^{(t)} + \alpha_{\min}\mathbf{D}^{(t)}$.

For criterion (4), the line search can be computed explicitly, since $J(\mathbf{C} + \alpha\mathbf{D})$ is a polynomial of oder 4 in α. More precisely, one has $J(\mathbf{C} + \alpha\mathbf{D}) = \sum_b j_b(\alpha)$, with $j_b(\alpha) = a_b(\alpha)^\dagger a_b(\alpha)$ where:

$$\begin{aligned} a_b(\alpha) &= V_{\exp_b} - (\mathbf{C}_{\ell(b)} + \alpha\mathbf{D}_{\ell(b)})^\dagger \beta_b (\mathbf{C}_{\ell(b)} + \alpha\mathbf{D}_{\ell(b)}) \\ &= V_{\exp_b} - \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)} - \alpha(\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)} + \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)}) + \alpha^2 \mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)} \end{aligned}$$

so that $j_b(\alpha) = p_b\alpha^4 + q_b\alpha^3 + r_b\alpha^2 + s_b\alpha + t_b$, with

$$\begin{aligned} p_b &= |\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)}|^2 \\ q_b &= -2\Re\Big((\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)})^\dagger (\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)} + \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)})\Big) \\ r_b &= |\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)} + \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)}|^2 + 2\Re\Big((\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)})^\dagger (V_{\exp_b} - \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)})\Big) \\ s_b &= -2\Re\Big((\mathbf{D}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)} + \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{D}_{k(b)})^\dagger (V_{\exp_b} - \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)})\Big) \\ t_b &= |V_{\exp_b} - \mathbf{C}_{\ell(b)}^\dagger \beta_b \mathbf{C}_{k(b)}|^2. \end{aligned}$$

and $J(\mathbf{C} + \alpha\mathbf{D}) = p\alpha^4 + q\alpha^3 + r\alpha^2 + s\alpha + t$, with $p = \sum_b p_b$, $q = \sum_b q_b$, $r = \sum_b r_b$, $s = \sum_b s_b$, $t = \sum_b t_b$. Thus, step 3 of the gradient algorithm can be performed this way:

1. compute p, q, r, s and t;
2. find the roots of $\frac{\partial J(\mathbf{C}+\alpha\mathbf{D})}{\partial\alpha} = 4p\alpha^3 + 3q\alpha^2 + 2r\alpha + s$;
3. among all real positive roots, select the one that gives the lowest value of $J(\mathbf{C}+\alpha\mathbf{D})$, and consequently update $\mathbf{C}$ and J.

In practice: Script `CG.m` *launches the optimisation. For gradient algorithm, set parameter* `algo` *to 1. Loading all quantities and initialisation are performed by script* `loadall.m`*. The 1D-minimisation (line search) is performed by function* `min1Dnew.m`*.*

1.3 Conjugate Gradient

The Conjugate Gradient (CG) algorithm follows the same principle as the gradient algorithm, except that the descent direction is chosen differently and does not correspond any more to the steepest descent. Indeed, for minimizing a *quadratic* criterion, the steepest descent algorithm (the gradient algorithm) approaches the solution asymptotically, whereas CG will find the solution in a finite number of iterations, which is the size of the problem. With CG, different descent directions (step 2 of the gradient algorithm) have been proposed, as:

- Polak-Ribière: $\mathbf{D}^{(t)} = -\overrightarrow{\nabla} J(\mathbf{C}^{(t)}) + \dfrac{\overrightarrow{\nabla} J(\mathbf{C}^{(t)})^{\dagger}(\overrightarrow{\nabla} J(\mathbf{C}^{(t)}) - \overrightarrow{\nabla} J(\mathbf{C}^{(t-1)}))}{\left\| \overrightarrow{\nabla} J(\mathbf{C}^{(t-1)}) \right\|^2} \mathbf{D}^{(t-1)}$

- Fletcher-Reeves: $\mathbf{D}^{(t)} = -\overrightarrow{\nabla} J(\mathbf{C}^{(t)}) + \dfrac{||\overrightarrow{\nabla} J(\mathbf{C}^{(t)})||^2}{||\overrightarrow{\nabla} J(\mathbf{C}^{(t-1)})||^2} \mathbf{D}^{(t-1)}$

- Conjugate descent: $\mathbf{D}^{(t)} = -\overrightarrow{\nabla} J(\mathbf{C}^{(t)}) + \dfrac{||\overrightarrow{\nabla} J(\mathbf{C}^{(t)})||^2}{\mathbf{D}^{(t-1)\dagger} \overrightarrow{\nabla} J(\mathbf{C}^{(t-1)})} \mathbf{D}^{(t-1)}$

The first one is recommended in [Bertsekas, 1995] since it shows more robustness toward the loss of conjugacy caused by non quadratic terms in the criterion and numerical errors brought by line search. Here the line search is explicit but the criterion is highly non-quadratic! Whatever the choice of the direction descent, thus, the conjugacy of the generated directions may be progressively lost. Thus, the CG optimisation procedures here operate in cycles of 100 CG steps, where each cycle begins with an iteration of the gradient algorithm. Note that this step also guarantees the convergence of the procedure.

In practice: Script `CG.m` *launches the optimisation. For CG algorithm, set parameter* `algo` *to 3.*

References

[Bertsekas, 1995] D.P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995.