BOOSTCONV

Paolo Luchini, Vincenzo Citro and Flavio Giannetti
CPLcode.net

January 22, 2022

Abstract

BoostConv is a subroutine that improves (or even enables) the convergence of a pre-existing, linear or nonlinear, iterative algorithm. Its reverse-calling interface only requires the insertion of a single line in the original code. The initial CPL implementation of BoostConv was written by Luchini in 2010, but remained private until a formal description of the algorithm, its translation to other programming languages and a number of its applications were published by Citro, Luchini, Giannetti and Auteri in 2017 [1].

1 Description of the algorithm

BoostConv is a subroutine that improves (or even enables) the convergence of a pre-existing, linear or nonlinear, iterative algorithm whose residual implicitly obeys the equation

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \mathbf{A}\mathbf{B}\boldsymbol{\xi}_n. \tag{1}$$

(See [1] for the derivation of eq. 14 there and the proof of its equivalence to a generic iterative algorithm.) Matrices $\mathbf{A}$ and $\mathbf{B}$, respectively representing the linearization of the equation to be solved and of its original feedback algorithm, need not be known to BoostConv. All that is needed is that they remain constant (for a linear problem) or slowly varying (for a nonlinear one) across iterations. In other words the same code block, however complicated, must repeat between calls to BoostConv. Under these conditions, only the insertion of a single subroutine call is required in the original code to boost its convergence, as will be shown by example.

Numerical vector $\boldsymbol{r}_n$ contains the set of residuals of the equations, to be driven to zero at convergence, and is the input parameter to BoostConv. Numerical vector $\boldsymbol{\xi}_n$ will be called "boosted residual", and is the modified residual output by

BoostConv in order to boost the convergence. When BoostConv is not in use, $\boldsymbol{\xi}_n = \boldsymbol{r}_n$.

The action of BoostConv is to update $\boldsymbol{r}_{n+1}$ in place with the corresponding value of $\boldsymbol{\xi}_{n+1}$, obtained from eq.(13) of [1]:

$$\boldsymbol{\xi}_n = \boldsymbol{r}_n + \sum_i c_i(\boldsymbol{u}_i - \boldsymbol{v}_i) \tag{2}$$

$\boldsymbol{u}_i$, $\boldsymbol{v}_i$ are a database of previous values of, respectively, $\boldsymbol{\xi}_n$, $\mathbf{AB}\boldsymbol{\xi}_n$, and provide an incomplete representation of matrix $\mathbf{AB}$. Coefficients c_i are calculated from a least-squares approximate inverse of $\mathbf{AB}$:

$$\left| \boldsymbol{r}_n - \sum_i c_i \boldsymbol{v}_i \right| = \min_{c_i};$$

$$\boldsymbol{u}_i = (\mathbf{AB})^{-1}\boldsymbol{v}_i \quad \text{(by definition).}$$

In this manner if the representation was complete, from (1) $\boldsymbol{r}_{n+1}$ would be identically zero. The database is built internally to BoostConv by accumulating historical values of pairs $\boldsymbol{\xi}_n$, $\boldsymbol{r}_n - \boldsymbol{r}_{n+1}$, which automatically obey the required relationship according to (1).

More than one strategy can be adopted in order to accumulate historical values, the only requirement being that all pairs must obey $\boldsymbol{v}_i = \mathbf{AB}\boldsymbol{u}_i$. One could be tempted to orthogonalize these pairs into an orthogonal basis, which could decrease roundoff error for a linear problem, but to do so becomes detrimental when the problem is nonlinear and older samples gradually become invalid because $\mathbf{AB}$ changes. If the database is orthogonalized, the original vectors become intermixed (replaced by linear combinations) and outdated information is never deleted, thus degrading the accuracy of the approximate inverse.

Since the ordering of basis vectors is irrelevant and their number is fixed, a strategy consists in specifying which vector pair must be deleted at each iteration and replaced by the new generated pair, and the simplest strategy is to replace the oldest. This strategy can be selected at the time of compilation by #defining DiscardOldest. A history longer than the number N of basis vectors can be covered by repeating more than one iteration of the basic algorithm before calling BostConv. A more elaborate, but equally simple to implement, strategy takes into account that, in the economy of allocating a small number N of costly basis vectors to the most efficient representation of history, a non-uniform sampling that increases with distance is probably preferable. This is the default. The best strategy will eventually have to be determined by trial and error, just as will the number N.

2 CPL implementation

In the BoostConv.cpl program the $\boldsymbol{v}$ vector is called `in(rot)`, and set on input to $\boldsymbol{r}_{n-1} - \boldsymbol{r}_n$, whereas the `out(rot)` vector is set to $\boldsymbol{u} - \boldsymbol{v} = \boldsymbol{\xi}$−`in(rot)`. The `rot` index rotates according to the strategy, and selects which `in` and `out` pair to operate upon at any given iteration. A static workspace is allocated to hold the vector basis and accompanying indices across iterations. Comments in the program provide further details of where each action takes place.

3 Usage

`BoostConv(ws,r,length)`

`ws` (optional) pointer to internal workspace of type `BoostConvWS`. If initially set to a NULL BoostConvWS, a new buffer will be internally allocated with size `length` and assigned to this pointer. Can be deallocated with FREE when no longer needed.

`r` on input, `REAL ARRAY` containing the residual vector of the iteration to be boosted; on output, substituted in place with the improved residual vector provided by the BoostConv algorithm.

`length` (optional) length of the history buffer. Only significant when ws points to a NULL BoostConvWS. Default: 10.

4 Example

See BoostConvGL.pdf.

References

[1] V. Citro, P. Luchini, F. Giannetti, and F. Auteri. Efficient stabilization and acceleration of numerical simulation of fluid flows by residual recombination. *Journal of Computational Physics*, 344:234–246, 2017. `doi:10.1016/j.jcp.2017.04.081`.