Ref-indic: invert details only where they are worth it

Reference manual

Version 1.5

Hend Ben Ameur François Clément Pierre Weis

2017-11-26

1 Outline

Ref-indic, Refinement indicators, is an adaptive parameterization platform using refinement indicators.

2 Description

The refinement indicator algorithm is suited for the estimation of a distributed parameter occurring in a mathematical simulation model, typically a set of partial differential equations. When the numerical simulation model must be solved on a fine grid, the refinement indicator algorithm provides an adaptive parameterization of the saught parameter that avoids over-parameterization difficulties. In each grid cell, the estimated parameter may be of dimension greater than one, i.e. the algorithm is able to estimate several scalar distributed parameters.

Ref-indic implements a generic version of the refinement indicator algorithm that can dock specific programs provided they conform to the generic algorithm API.

The API of Ref-indic requires four main functionalities (called tasks) for the user specific program, it must be able:

- to initialize, i.e. to open all necessary data files, to perform all necessary preliminary computation, and to return an initial coarse parameterization (giving a zone number between 0 and the initial number of zones minus one for each cell of the fine grid);
- to compute the gradient on the fine grid for a given fine parameterization;
- to optimize the problem for a given coarse parameterization;
- and to finalize, i.e. to store the resulting coarse parameterization.

Given any such user specific program, the inversion platform automatically provides a program that solves the corresponding user inverse problem using the refinement indicator algorithm.

In its current implementation, the inversion platform can only build coarse parameterizations for a distributed parameter defined on a fine rectangular grid. From version 1.5+pl0, the user has the possibility to specify masked cells in the fine rectangular grid that will be ignored by the algorithm (with the use of the specific zone number -1 in the initial coarse parameterization). This allows for the treatment of inverse problems defined on unstructured meshes. The handling of both-way interpolations must be taken care of by the gradient computation and optimization tasks. The masked cells must be the same for all components of the parameter.

3 Installation of Ref-indic

See also the `INSTALL` file in the root directory of the distribution.

3.1 Requirements

You need OCaml version 4.01 or higher to build Ref-indic.

The OCaml image library Grimage, the polyglot Input/Output library Pio, the OCaml user's additional library Ulib, and the OCaml module Generator OCamlGen are also required to build Ref-indic. They are provided, and their configuration, compilation, and installation (in a private location) are fully automatic.

Configuration, compilation and installation uses Pph (Portable Project Handler) technology. It is provided, and its use is fully automatic.

3.2 Installation instructions

To configure, build and install Ref-indic, type

```
./configure [-ird <installation_root_directory>]
make
make install   # Note that you may need root privileges for that.
```

When called without the `-ird` option, the `configure` script computes the root directory for the installation from the system default value, or from previous run of the same script.

Other options for the configuration phase may be accessible, type

```
./configure -help
```

To remove all files installed by `make install`, all files built by `make`, and all files created by `./configure`, respectively type

```
make uninstall   # Note that you may also need root privileges for that.
make clean
./unconfigure
```

To uninstall all versions of Ref-indic, type

```
make uninstall-all-versions   # Note that you may also need root privileges
                              # for that.
```

4 Examples

Examples are provided in the `example` directory. They are all configured to run the inversion platform in interactive mode. Thus, it is best to run them separately in each sub-directory (see each dedicated `README` file):

- `identity` (OCaml and Fortran implementations);
- `transmissivity` (Fortran implementation);
- `hydrogeology` (Fortran implementation).

To configure (set up and compile), and run examples of Ref-indic, respectively type

```
make configure-example
make example
```

To remove all files created when running the examples, and those built when configuring them, respectively type

```
make clean-example
make unconfigure-example
```

Note that all examples do not need that Ref-indic is installed to run properly.

5 Support

Please send bug reports to `refinement-bugs@inria.fr`, comments to `refinement-devel@inria.fr`, and contact the user community at `refinement-users@inria.fr`.

6 License

Ref-indic is free software distributed under the 3-clause BSD license. See the `LICENSE` file for details.

7 Acknowledgements

The algorithm implemented in Ref-indic comes from papers co-written with Guy Chavent.

Ref-indic is entirely written in OCaml.

8 What is adaptive parameterization?

Ref-indic implements the adaptive parameterization algorithm using refinement indicators to solve inverse problems set as minimization problems of the form

$$\arg\min_p J(p) \stackrel{\text{def}}{=} \frac{1}{2}\|d - \mathcal{F}(p)\|^2,$$

where $\mathcal{F}$ models the cause-to-effect relation from parameter p to measures d (it is the function to invert). Typically, computing measures $\mathcal{F}(p)$ for some parameter p involves the resolution of a set of PDEs depending on the distributed parameter p (e.g., a function of the space variable).

Parameter p is searched for under the form $p = \mathcal{P}m$ where $\mathcal{P}$ is a piecewise constant parameterization operator splitting the domain of function p into a given number of—constant—pieces; m is called *coarse parameter* (one value per piece), and in contrast, p is now called *fine parameter* (one value per point). Piecewise constant parameterization operators are fully and uniquely defined by the choice of a partition of the space domain (with no empty part), hence the identification of partitions and parameterization operators.

To avoid over-parameterization problems, pieces are progressively split one at a time in an optimal manner to build an optimal sequence of parameterization operators $(\mathcal{P}_n)_{n=1,2,\ldots}$ where each $\mathcal{P}_n$ is associated with a partition into n pieces. Hence the name "adaptive parameterization algorithm".

The approach is efficient for inverse problems in which modeling operator $\mathcal{F}$ is almost linear with respect to low frequency contents of the parameter p, and may become more and more nonlinear as the frequency contents increase. It is not suited at all to the case where low frequencies of the unknown parameter are related to a highly nonlinear behavior of the model, as for seismic inversion.

9 Algorithmic details

Input data are a dimension management (`Vector`, or `Best_component_only`), a refinement strategy (`Overall_best`, or `Best_in_a_family`), and an initial vector partition $\mathcal{P}_1$. An user-supplied external program provides the following functions:

$$\texttt{optim}: \ \mathcal{P} \longmapsto (m^\star, J^\star) = (\arg\min_m J(\mathcal{P}m), J(\mathcal{P}m^\star)), \quad \texttt{grad}: \ p' \longmapsto \nabla_p J(p').$$

For `Vector` dimension management, all components are always subject to the same scalar partition. For `Best_component_only` dimension management, different components may be subject to different scalar partitions.

A *cutting* $c_{\mathbf{P}}$ splits some part $\mathbf{P}$ of some partition $\mathcal{P}$ into nonempty disjoint subparts $\mathbf{P}^+$ and $\mathbf{P}^-$.

Initialization

0. Compute optimal parameter associated with initial vector partition $\mathcal{P}_1$,

$$(m_1, J_1) = \texttt{optim}(\mathcal{P}_1).$$

Iterations For $n \geq 1$, until stopping criterion is satisfied, do:

1. Compute fine gradient, $g_n = \mathtt{grad}(\mathcal{P}_n m_n)$.

2a. Compute scalar first order indicator for all scalar cuttings in all parts for all components,

$$\forall k,\ \forall \mathbf{P} \in \mathcal{P}_n^k,\ \forall c_{\mathbf{P}} = (\mathbf{P}^+, \mathbf{P}^-) \in C_{\mathbf{P}}, \quad I_{c_{\mathbf{P}}}^k = \left| \sum_{i \in \mathbf{P}^+} (g_n^k)_i - \sum_{i \in \mathbf{P}^-} (g_n^k)_i \right|,$$

2b. For `Vector` dimension management, compute vector first order indicator for all vector cuttings in all parts,

$$\forall \mathbf{P} \in \mathcal{P}_n,\ \forall c_{\mathbf{P}} \in C_{\mathbf{P}}, \quad I_{c_{\mathbf{P}}} = \|(I_{c_{\mathbf{P}}}^k)_k\|_q.$$

With the refinement strategy `Overall_best`, we have to take $q = \infty$ when the dimension of the parameter is greater than 1.

3. Select best candidate cuttings (highest first order indicators),
`Vector`: $C^\star \subset \bigcup_{\mathbf{P} \in \mathcal{P}_n} C_{\mathbf{P}}$, `Best_component_only`: $C^\star \subset \bigcup_k \bigcup_{\mathbf{P} \in \mathcal{P}_n^k} C_{\mathbf{P}}$.

4. Compute exact indicator for all selected cuttings,

$$\forall c^\star \in C^\star, \quad (m_{c^\star}, J_{c^\star}) = \mathtt{optim}(\mathcal{P}_{c^\star}).$$

5. Pick best selected cutting (lowest exact indicator): $c^{\text{opt}} = \arg\min_{c^\star} J_{c^\star}$

$$(\mathcal{P}_{n+1}, m_{n+1}, J_{n+1}) = (\mathcal{P}_{c^{\text{opt}}}, m_{c^{\text{opt}}}, J_{c^{\text{opt}}}).$$

References

[1] H. Ben Ameur, G. Chavent, F. Clément, and P. Weis. Image segmentation with multidimensional refinement indicators. *Inverse Problems in Science and Engineering*, 19(5):577–597, 2011. Special Issue: Proceedings of the 5th Internat. Conf. on Inverse Problems: Modeling and Simulation, May 24th–29th, 2010, held in Antalya, Turkey.

[2] H. Ben Ameur, F. Clément, P. Weis, and G. Chavent. The multidimensional refinement indicators algorithm for optimal parameterization. *Journal of Inverse and Ill-Posed Problems*, 16(2):107–126, 2008.