TaylUR 3, a multivariate arbitrary-order automatic differentiation package for Fortran 95 *

G.M. von Hippel

NIC, DESY, Platanenallee 6, 15738 Zeuthen, Germany

A R T I C L E  I N F O

Article history:
Received 22 October 2009
Accepted 26 October 2009
Available online 28 October 2009

MSC:
41A58
41-04
65D25

Keywords:
Automatic differentiation
Higher derivatives
Fortran 95

A B S T R A C T

This new version of TaylUR is based on a completely new core, which now is able to compute the numerical values of all of a complex-valued function’s partial derivatives up to an arbitrary order, including mixed partial derivatives.

New version program summary

Program title: TaylUR
Catalogue identifier: ADXR_v3.0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADXR_v3_0.html
Program obtainable from: CPC Program Library, Queen’s University, Belfast, N. Ireland
Licensing provisions: GPLv2

Nature of problem:
Problems that require potentially high orders of partial derivatives with respect to several variables or derivatives of complex-valued functions, such as e.g. momentum or mass expansions of Feynman diagrams in perturbative QFT, and which previous versions of this TaylUR [1,2] cannot handle due to their lack of support for mixed partial derivatives.

Solution method:
Arithmetic operators and Fortran intrinsics are overloaded to act correctly on objects of a defined type taylor, which encodes a function along with its first few partial derivatives with respect to the user-defined independent variables. Derivatives of products and composite functions are computed using multivariate forms [3] of Leibniz’s rule $D^\nu (fg) = \sum_{\mu \leq \nu} \frac{\mu!}{\mu!(\nu-\mu)!} D^\mu f D^{\nu-\mu} g$ where $\nu = (\nu_1, \ldots, \nu_d)$, $|\nu| = \sum_{j=1}^d \nu_j$, $v! = \prod_{j=1}^d v_j!$, $D^\nu f = \partial^{\nu_1} f / (\partial x_1)^{\nu_1} \cdots \partial^{\nu_d} f / (\partial x_d)^{\nu_d}$, and $\mu \leq \nu$ if either $|\mu| \leq |\nu|$ or $|\mu| = |\nu|$ but $\mu_i \leq \nu_i$ for some $i \in \{0, \ldots, d-1\}$, and of Faa di Bruno’s formula $D^\nu (f \circ g) = \sum_{|\mu| = \nu} \frac{\nu!}{\mu!} \sum_{\sum_{i=1}^d s_i = \nu} v! / \prod_{i=1}^d (\prod_{j=1}^{s_i} k_j ! (\nu_j !)^{s_i} (\mu_j !)^{s_i}) (D^{\mu_j} g)^{s_i}$ where the sum is over $\{k_1, \ldots, k_d; \lambda_1, \ldots, \lambda_d \} \in \mathbb{Z}^{d+1}_{\geq 0}$, $\sum_{i=1}^d k_i > 0$, $0 < \lambda_1 < \cdots < \lambda_d$, $\sum_{i=1}^d k_i = p$, $\sum_{i=1}^d k_i \lambda_i = \nu$. An indexed storage system is used to store the higher-order derivative tensors in a one-dimensional array. The relevant indices $(k_1, \ldots, k_d; \lambda_1, \ldots, \lambda_d)$ and the weights occurring in the sums in Leibniz’s and Faa di Bruno’s formula are precomputed at startup and stored in static arrays for later use.

Reasons for new version:
The earlier version lacked support for mixed partial derivatives, but a number of projects of interest required them.

Summary of revisions:
The internal representation of a taylor object has changed to a one-dimensional array which contains the partial derivatives in ascending order, and in lexicographic order of the

* This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
E-mail address: georg.von.hippel@desy.de.
URL: http://www-zeuthen.desy.de/~hippel.
0010-4655/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
corresponding multiindex within the same order. The necessary mappings between multiindices and indices into the taylor objects' internal array are computed at startup. To support the change to a genuinely multivariate taylor type, the DERIVATIVE function is now implemented via an interface that accepts both the older format derivative(f, mu(n)) = ∂^μ f and also a new format derivative(f, mu(:,)) = D^μ f that allows access to mixed partial derivatives. Another related extension to the functionality of the module is the HESSIAN function that returns the Hessian matrix of second derivatives of its argument. Since the calculation of all mixed partial derivatives can be very costly, and in many cases only some subset is actually needed, a masking facility has been added. Calling the subroutine DEACTIVATE_DERIVATIVE with a multiindex as an argument will deactivate the calculation of the partial derivative belonging to that multiindex, and of all partial derivatives it can feed into. Similarly, calling the subroutine ACTIVATE_DERIVATIVE will activate the calculation of the partial derivative belonging to its argument, and of all partial derivatives that can feed into it. Moreover, it is possible to turn off the computation of mixed derivatives altogether by setting Diagonal_taylors to .TRUE.. It should be noted that any change of Diagonal_taylors or Taylor_order invalidates all existing taylor objects. To aid the better integration of TaylUR into the HPSrc library [4], routines SET_DERIVATIVE and SET_ALL_DERIVATIVES are provided as a means of manually constructing a taylor object with given derivatives.

Restrictions: Memory and CPU time constraints may restrict the number of variables and Taylor expansion order that can be achieved. Loss of numerical accuracy due to cancellation may become an issue at very high orders.

Unusual features: These are the same as in previous versions, but are enumerated again here for clarity. The complex conjugation operation assumes all independent variables to be real. The functions REAL and AIMAG do not convert to real type, but return a result of type taylor (with the real/imaginary part of each derivative taken) instead. The user-defined functions VALUE, REALVALUE and IMAGVALUE, which return the value of a taylor object as a complex number, and the real and imaginary part of this value, respectively, as a real number are also provided. Fortran 95 intrinsics that are defined only for arguments of real type (ACOS, AINT, ANINT, ASIN, ATAN, ATAN2, CEILING, DIM, FLOOR, INT, LOG10, MAX, MAXLOC, MAXVAL, MIN, MINLOC, MINVAL, MOD, MODULO, NINT, SIGN) will silently take the real part of taylor-valued arguments unless the module variable Real_args_warn is set to .TRUE.., in which case they will return a quiet NaN value (if supported by the compiler) when called with a taylor argument whose imaginary part exceeds the module variable Real_args_tol. In those cases where the derivative of a function becomes undefined at certain points (as for ABS, AINT, ANINT, MAX, MIN, MOD, and MODULO), while the value is well defined, the derivative fields will be filled with quiet NaN values (if supported by the compiler).

Running time: The running time of TaylUR operations grows rapidly with both the number of variables and the Taylor expansion order. Judicious use of the masking facility to drop unneeded higher derivatives can lead to significant accelerations, as can activation of the Diagonal_taylors variable whenever mixed partial derivatives are not needed.

Acknowledgments: The author thanks Alistair Hart for helpful comments and suggestions. This work is supported by the Deutsche Forschungsgemeinschaft in the SFB/TR 09.

References:

© 2009 Elsevier B.V. All rights reserved.