Higher Order Univariate AD

Object is one vector with $(n+1)$ entries, where n is the order.
Coefficients of vector are Taylor coefficients

$$a_i = \frac{1}{i!}\frac{\partial^i f}{\partial x^i}$$

(Could also use derivatives directly, but with Taylor coefficients, subsequent arithmetic is simpler)

Addition: merely component-wise.
Multiplication:

$$c_i = \sum_{j=0}^{i} a_j \cdot b_{i-j}$$

Very straightforward, easy to program.

Higher Order Univariate AD - Intrinsics

Various methods. For example,

$$
\begin{aligned}
\sin\left((a_0, a_1, \ldots, a_n)\right) &= \sin\left((a_0, 0, \ldots, 0) + (0, a_1, a_2, \ldots, a_n)\right) \\
&= \sin(a_0, 0, \ldots, 0) \cdot \cos(0, a_1, a_2, \ldots, a_n) + \cos(a_0, 0, \ldots, 0) \cdot \sin(0, a_1, \\
&= (\sin(a_0), 0, \ldots, 0) \cdot \sum_{i=1}^{\infty} \frac{(-1)^i}{(2i)!}(0, a_1, a_2, \ldots, a_n)^{2i} \\
&+ (\cos(a_0), 0, \ldots, 0) \cdot \sum_{i=1}^{\infty} \frac{(-1)^i}{(2i+1)!}(0, a_1, a_2, \ldots, a_n)^{2i+1} \\
&= (\sin(a_0), 0, \ldots, 0) \cdot \sum_{i=1}^{[n/2]} \frac{(-1)^i}{(2i)!}(0, a_1, a_2, \ldots, a_n)^{2i} \\
&+ (\cos(a_0), 0, \ldots, 0) \cdot \sum_{i=1}^{[n/2]} \frac{(-1)^i}{(2i+1)!}(0, a_1, a_2, \ldots, a_n)^{2i+1}
\end{aligned}
$$

Higher Order Univariate AD - Intrinsics

Thus, used addition theorem to split off constant part a, use (finite) power series for non-constant part b.

Can be used in many cases:

$$\cos(a+b) = \cos(a)\cdot\cos(b) - \sin(a)\cdot\sin(b)$$
$$\exp(a+b) = \exp(a)\cdot\exp(b)$$
$$\log(a+b) = \log(a) + \log\left(1+\frac{b}{a}\right) \text{ if } a \neq 0$$
$$\frac{1}{(a+b)} = \frac{1}{a}\cdot\left(1+\frac{b}{a}\right)^{-1} \text{ if } a \neq 0$$
$$\sqrt{a+b} = \sqrt{a}\cdot\sqrt{1+\frac{b}{a}} \text{ if } a \neq 0$$

etc etc

The required sums usually have n terms.

Other method: Use **Newton method** with zeroth-order solution as start value. This usually requires less iterations, namely about $\log_2(n)$

Higher Order Coefficient Combinatorics

How many monomials are there up to order n in v variables? Write them as

$$x_1^{i_1} * x_2^{i_2} * x_3^{i_3} * ... * x_v^{i_v}$$

with $i_1 + ... + i_v \leq n$. Consider $x_1^2 * x_2^3 * x_3 \; * ... * x_v^3$. Code it as

$$\overbrace{11}^{i_1=2} * \overbrace{111}^{i_2=3} * \overbrace{1}^{i_3=1} * ... * \overbrace{111}^{i_v=3} * \overbrace{111}^{n-i_1-...-i_v}$$

Each monomial is uniquely represented in such a way. Observe total number of $1'$s is n, total number of $*'s$ is v. So, total length of string is $(n + v)$.

The placement of the $*'$s determines everything. Apparently there are

$$N(n, v) = \binom{n+v}{v}$$

ways to arrange them.

Higher Order Coefficient Combinatorics - More

How many monomials are there of *exact* order n? Code them as

$$\overbrace{11}^{i_1=2} * \overbrace{111}^{i_2=3} * \overbrace{1}^{i_3=1} * \ldots * \overbrace{111}^{i_v=3}$$

Thus there are

$$\binom{n+v-1}{v-1}$$

Number of possible products of two monomials of total order up to n? Code them as

$$\overbrace{11}^{i_1=2} * \ldots * \overbrace{111}^{i_v=3} * \overbrace{111}^{j_1=3} * \ldots * \overbrace{1111}^{i_v=4} * \overbrace{11}^{n-i_1-\ldots-i_v-j_1-\ldots-j_v}$$

Length of string is $(n+2v)$, and the number of $*$'s is $2v$. Thus number of possible products is

$$\binom{n+2v}{2v}.$$

Multivariate from Univariate AD (Griewank, 1992)

Idea: compute many univariate derivatives in different "directions", determine the higher mixed partials from linear algebra.

$$f(x,y) = c + (b_1, b_2) \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \frac{1}{2}(x,y) \cdot \begin{pmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Want all partial derivatives up to order 2. Determine derivatives in x-direction (directional derivatives in direction $(1,0)$)

$$(c, b_1, h_{11})$$

Derivatives in y-direction (directional derivatives in direction $(0,1)$)

$$(c, b_2, h_{22})$$

Directional derivatives in direction $d = (1,1)$. We have

$$f(hd) = c + h(b_1 + b_2) + \frac{1}{2}h^2(h_{11} + h_{12} + h_{12} + h_{22}),$$

so directional derivative has form

$$(c, b_1 + b_2, h_{11} + 2h_{12} + h_{22})$$

Thus, we can reconstruct all partials up to order 2.

Multivariate from Univariate AD - Properties

Higher orders and more variables: Linear algebra needed to obtain mixed partials from suitable univariate directional partials

Advantages:

- Conceptually simple - requires only univariate AD
- No complicated addressing schemes

Possible Limitations:

- Requires multiple passes
- Linear Algebra becomes potentially unwieldy, ill-conditioned
- Can not easily accommodate sparsity in derivatives, i.e. treat only the nonzero ones (see below)
- Can not be used with Taylor models (see below)

Multivariate AD - Direct Method (Berz, 1985)

Idea: accumulate all Taylor coefficients simultaneously.
Key Problem: Determine a particular arrangement of all derivatives.

Advantages:

- Can be combined with sparsity treatment
- No need for repeated sweeps
- No need for linear algebra
- If done right, requires fewer operations
- Can be used with Taylor models (see below)

Possible Limitations:

- Requires sophisticated addressing scheme for multiplication
- Efficiency limited by that of multiplication

Storage and Addition

Each nonzero derivative is represented by its Taylor coefficients and several coding integers

$$c_i,\ n_{1,i}\ n_{2,i}, \ldots$$

More about the meaning of coding integers n later. All these are **sorted**, first by value of n_1, and then by value of n_2, etc

Addition: Go through both sorted lists with pointers p_1 and p_2

- If coding integers of both lists agree, add coefficients.
  - If result is zero, increment p_1 and p_2.
  - f result is nonzero, increment pointer of result p_r, copy coefficient sum and coding integers, increment p_1 and p_2.
- If coding integers do not agree, copy term with integers that come first, increment its pointer, and p_r

Multiplication

Given two Taylor polynomials with coefficients (a_i) and (b_i) and exponents $(n_{i,j})$ and $(m_{i,j})$. Suppose the $N = (n+v)!/n!/v!$ monomials are arranged in a certain order in vector of derivatives.

Most intuitive way:

Let M_i: monomial stored in ith component, and let I_M denote the position of the monomial M.

Coefficient of component i of result is given by

$$c_i = \sum_{\substack{0 \le \nu, \mu \le N \\ M_\nu \cdot M_\mu = M_i}} a_\nu \cdot b_\mu.$$

But: very difficult to determine all contributing factors M_ν and M_μ with $M = M_\nu \cdot M\mu$ online. Even if they are pre-stored for every i,this can not easily take care of sparsity.

Better way:

Multiply each nonzero monomial in first vector with all those nonzero monomials in the second vector.

Naturally avoids vanishing coefficients.

Multiplication - Single Stage Coding

Let $M = x_1^{i_1} \cdot ... \cdot x_v^{i_v}$, then $n_c(M)$ is defined as follows:

$$\begin{aligned} n_c(M) &= n_c(x_1^{i_1} \cdot ... \cdot x_v^{i_v}) \\ &= i_1 \cdot (n+1)^0 + i_2 \cdot (n+1)^1 + ... + i_v \cdot (n+1)^{v-1}. \end{aligned}$$

So exponents become "digits" in a base $(n+1)$ representation. Since $i_\nu \leq n$, the function $M \to n(M)$ is injective and hence the **coding is unique.**

No coding exceeds $(n+1)^v$, but not all such codings occur.

Now want to multiply two monomials M and N and retain terms less than order n. Since multiplication corresponds to addition of the exponents, it follows that

$$n_c(M \cdot N) = n_c(M) + n_c(N).$$

To find of the desired coordinate position I_M of the product of two monomials, need a **lookup array** p that has the property

$$I_M = p(n_c(M)).$$

Has to be computed only once for given order n and number of variables v.

Disadvantage: since codings are bounded by $(n+1)^v$, the array needs to have at least this length. $n = 9$, $v = 10$ leads to length 10^{10}.

Multiplication - Multi-Stage Coding

Two-Stage Coding: Define two coding integers

$$\begin{aligned} n_1(x_1^{i_1}\cdot \dots \cdot x_v^{i_v}) =& i_1 \cdot (n+1)^0 + i_2 \cdot (n+1)^1 \\ &+ \cdots + i_{\frac{v}{2}} \cdot (n+1)^{(\frac{v}{2}-1)} \\ n_2(x_1^{i_1}\cdot \dots \cdot x_v^{i_v}) =& i_{\frac{v}{2}+1} \cdot (n+1)^0 + i_{\frac{v}{2}+2} \cdot (n+1)^1 \\ &+ \cdots + i_v \cdot (n+1)^{(\frac{v}{2}-1)}. \end{aligned}$$

Sort the $N(n,v)$ monomials first by value of n_1, and then by value of n_2. Observe that

$$\begin{aligned} n_1(M \cdot N) &= n_1(M) + n_1(N) \\ n_2(M \cdot N) &= n_2(M) + n_2(N). \end{aligned}$$

Introduce "inverse" arrays p_1 and p_2 in the following way: For all n_1 and n_2 that appear as valid coding integers, we set

$p_1(n_1) = (I_M$ of first monomial M with first coding integer $n_1)$ and

$p_2(n_2) = (I_M$ of first monomial M with second coding integer $n_2) - 1$.

Again p_1 and p_2 can be generated once during initial setup process. Can now calculate address of product as

$$I_{M\cdot N} = p_1[n_1(I_M) + n_1(I_N)] + p_2[n_2(I_M) + n_2(I_N)].$$

Multiplication - Multi-Stage Coding, Analysis

In two-stage coding, each address computation requires three integer additions and two array lookups.

Advantage: Storage needed for p_1 and p_2 is now only $(n+1)^{v/2}$. For example of $n = 9$, $v = 10$ leads to length 10^5.

Can be generalized to $s > 2$ stages: exponents grouped into s blocks, and arranged such that block s takes precedence over block $s-1$, which takes precedence over that of block $s-2$, etc.

Each monomial is assigned s coding integers $n_1 ... n_s$, and there are s "inverse" arrays $p_1 ... p_s$. Address computation:

$$I_{M \cdot N} = p_1[n_1(I_M)+n_1(I_N)]+p_2[n_2(I_M)+n_2(I_N)]+...+p_x[n_x(I_M)+n_x(I_N)]$$

Required storage: only $(n+1)^{\frac{v}{2}}$.

Maximally compact, and maximally costly, at $s = v$: Storage for reverse array only $(n+1)$, but $v+1$ integer additions

Practically Relevance: For most problems, two stage scheme is optimal because of intrinsic limitations due to cost of (dense) multiplication

$$\binom{n+2v}{2v}.$$

Multiplication - Two Stage Coding, Example

Consider case $n = 3$, $v = 4$

#	I1	I2	I3	I4	ORDER	N1	N2
1	0	0	0	0	0	0	0
2	1	0	0	0	1	1	0
3	0	1	0	0	1	4	0
4	2	0	0	0	2	2	0
5	1	1	0	0	2	5	0
6	0	2	0	0	2	8	0
7	3	0	0	0	3	3	0
8	2	1	0	0	3	6	0
9	1	2	0	0	3	9	0
10	0	3	0	0	3	12	0
11	0	0	1	0	1	0	1
12	1	0	1	0	2	1	1
13	0	1	1	0	2	4	1
14	2	0	1	0	3	2	1
15	1	1	1	0	3	5	1
16	0	2	1	0	3	8	1

#	P1	P2
0	1	0
1	2	10
2	4	22
3	7	31
4	3	16
5	5	25
6	8	32
7	0	0
8	6	28
9	9	33
10	0	0
11	0	0
12	10	34

17	0	0	0	1	1	0	4
18	1	0	0	1	2	1	4
19	0	1	0	1	2	4	4
20	2	0	0	1	3	2	4
21	1	1	0	1	3	5	4
22	0	2	0	1	3	8	4
23	0	0	2	0	2	0	2
24	1	0	2	0	3	1	2
25	0	1	2	0	3	4	2
26	0	0	1	1	2	0	5
27	1	0	1	1	3	1	5
28	0	1	1	1	3	4	5
29	0	0	0	2	2	0	8
30	1	0	0	2	3	1	8
31	0	1	0	2	3	4	8
32	0	0	3	0	3	0	3
33	0	0	2	1	3	0	6
34	0	0	1	2	3	0	9
35	0	0	0	3	3	0	12

Multiplication - Weighting

Sometimes important: Carry different variables x_i to different orders w_i.

Can be achieved by simply "seeding" original variables as

$$P(x) = (x_1^{w_1}, x_2^{w_2}, ..., x_v^{w_v}).$$

Then in all subsequent operations, only multiples of w_i appear as powers of x_i. Optimal reduction of speed by sparsity, but suboptimal memory use. Use weighted coding:

$$\begin{aligned}
n_1(x_1^{i_1} \cdot \cdots \cdot x_v^{i_v}) =& \frac{i_1}{w_1} + \frac{i_2}{w_2} \cdot \left(\left[\frac{n}{w_1} \right] + 1 \right) + \frac{i_3}{w_3} \cdot \left(\left[\frac{n}{w_1} \right] + 1 \right) \cdot \left(\left[\frac{n}{w_2} \right] + 1 \right) \\
& + \cdots + \frac{i_{\frac{v}{2}}}{w_{\frac{v}{2}}} \cdot \prod_{k=1}^{\frac{v}{2}-1} \left(\left[\frac{n}{w_k} \right] + 1 \right) \\
n_2(x_1^{i_1} \cdot \cdots \cdot x_v^{i_v}) =& \frac{i_{\frac{v}{2}+1}}{w_{\frac{v}{2}+1}} + \cdots + \frac{i_v}{w_v} \cdot \prod_{k=\frac{v}{2}+1}^{v-1} \left(\left[\frac{n}{w_k} \right] + 1 \right).
\end{aligned}$$

"[ ]": Gauss bracket. So, exponents are divided by their weighting factor, and resulting quotients are "digits" in a "variable-base" representation.

Multiplication - Weighting, Example

Consider case $n = 5$, $v = 3$. Tables without weighting:

```
  j  i1 i2 i3     n1    n2   Order        n     p1   p2
 **********************************      **************
  1   0  0  0      0     0     0          0      1    0
  2   1  0  0      1     0     1          1      2   21
  3   0  1  0      6     0     1          2      4   36
  4   2  0  0      2     0     2          3      7   46
  5   1  1  0      7     0     2          4     11   52
  6   0  2  0     12     0     2          5     16   55
         ... continues ...                .. continues ..
 53   0  0  4      0     4     4         28      0    0
 54   1  0  4      1     4     5         29      0    0
 55   0  1  4      6     4     5         30     21    0
 56   0  0  5      0     5     5
```

There are 56 monomials, and the reverse addressing arrays need to have at least length 30.

Multiplication - Weighting, Example

Now consider weighting $w_1 = 5$, $w_2 = 1$, $w_3 = 2$. Again we have

$$n_1(M \cdot N) = n_1(M) + n_1(N),\ \ n_2(M \cdot N) = n_2(M) + n_2(N).$$

j	i1	i2	i3	n1	n2	Order
1	0	0	0	0	0	0
2	0	1	0	2	0	1
3	0	2	0	4	0	2
4	0	3	0	6	0	3
5	0	4	0	8	0	4
6	5	0	0	1	0	5
7	0	5	0	10	0	5
8	0	0	2	0	1	2
9	0	1	2	2	1	3
10	0	2	2	4	1	4
11	0	3	2	6	1	5
12	0	0	4	0	2	4
13	0	1	4	2	2	5

n	p1	p2
0	1	0
1	6	7
2	2	11
3	0	0
4	3	0
5	0	0
6	4	0
7	0	0
8	5	0
9	0	0
10	7	0

Multiplication - Weighting, Example

Examples for storage costs. Consider various choices for n and v, and different weighting.

In all examples, $w_1 = 1$, and other weights are w.

dim	order	max number of monomials			size of the inverse integer lists		
v	n	no weighting	$w=3$	$w=5$	no weighting	$w=3$	$w=5$
8	10	43758	375	81	13310	640	270
8	12	125970	825	153	26364	1500	324
8	18	1562275	5577	705	123462	6174	1152
10	10	184756	638	110	146410	2560	810
10	12	646646	1573	220	342732	7500	972
10	18	13123110	14014	1210	2345778	43218	4608
12	10	646646	1001	143	1610510	10240	2430
12	12	2704156	2730	299	4455516	37500	2916
12	18	86493225	30940	1911	47045880	302526	18432

Checkpointing and Composition

Reverse AD thrives from the fact that dependence of later intermediate variables on earlier intermediate variables is very sparse. See "cheap gradient theorem" etc etc.

How can this be used for higher order AD?

1. Compute high-order dependence of suitable subsequent intermediates on earlier intermediates. Will exhibit similar sparsity as in first order case.

2. Patch together such dependencies through composition.

Composition operation: Let us assume multivariate functions f at some point x_0 has Taylor polynomial P_f, and multivariate function g at $f(x_0)$ has Taylor polynomial P_g. Then, Taylor polynomial of $g \circ f$ is obtained from

$$P_{g \circ f} = P_g(P_f),$$

i.e. by evaluating the known Taylor polynomial of g with the "seed" P_f.

Can often be used very beneficially to perform side calculations to lower dimension. For example, in Beam Physics, motion computation has $v = 6$, but field computation has $v = 3$.

COSY

Design Features:

1. Uses two-stage coding, sparse storage of derivatives
2. All standard intrinsics as well as Derivation, Antiderivation
3. Highly optimized implementation
4. Can be called from F77 and C (subroutine calls), F95 and C++ (objects)
5. Language-Independent Platform - only one source code for four languages
6. Altogether nearly 1000 registered users, development almost 20 years, $5M in funding

Existing Application Packages:

1. COSY INFINITY (Beam Physics): Currently the main tool for simulation of nonlinear high-order effects in beam dynamics
2. COSY-VI: Validated Integrator, based on Taylor expansion in time AND initial condition
3. COSY-GO: Validated Global Optimizer, based on Taylor expansion for dependency suppression and domain reduction

Definitions - Taylor Models and Operations

We begin with a review of the definitions of the basic operations.

Definition (Taylor Model) Let $f : D \subset R^v \to R$ be a function that is $(n+1)$ times continuously partially differentiable on an open set containing the domain v-dimensional domain D. Let x_0 be a point in D and P the n-th order Taylor polynomial of f around x_0. Let I be an interval such that

$$f(x) \in P(x - x_0) + I \text{ for all } x \in D.$$

Then we call the pair (P, I) an n-th order Taylor model of f around x_0 on D.

Definition (Addition and Multiplication) Let $T_{1,2} = (P_{1,2}, I_{1,2})$ be n-th order Taylor models around x_0 over the domain D. We define

$$T_1 + T_2 = (P_1 + P_2, I_1 + I_2)$$
$$T_1 \cdot T_2 = (P_{1\cdot 2}, I_{1\cdot 2})$$

where $P_{1\cdot 2}$ is the part of the polynomial $P_1 \cdot P_2$ up to order n and

$$I_{1\cdot 2} = B(P_e) + B(P_1) \cdot I_2 + B(P_2) \cdot I_1 + I_1 \cdot I_2$$

where P_e is the part of the polynomial $P_1 \cdot P_2$ of orders $(n+1)$ to $2n$, and $B(P)$ denotes a bound of P on the domain D. We demand that $B(P)$ is at least as sharp as direct interval evaluation of $P(x - x_0)$ on D.

Implementation of TM Arithmetic

Validated Implementation of TM Arithmetic exists. The following points are important

- Strict requirements for **underlying FP arithmetic**
- Taylor models require cutoff threshold (**garbage collection**)
- Coefficients remain FP, not intervals
- Package quite **extensively tested** by Corliss et al.

For practical considerations, the following is important:

- Need **sparsity** support
- Need efficient coefficient **addressing** scheme
- About $50,000$ lines of code
- **Language Independent** Platform, coexistence in F77, C, F90, C++

Efficient Taylor Models - Sign Choice

Decompose polynomials to multiply into purely positive one and purely negative one:

$$P_{1,2} = P_{1,2}^{+} + P_{1,2}^{-}$$

where all coefficients in $P_{1,2}^{+}$ are positive, and all coefficients in $P_{1,2}^{-}$ are negative. Then execute separately

$$\begin{aligned} Q^{+} &= P_1^{+} \cdot P_2^{+} + P_1^{-} \cdot P_2^{-} \text{ and} \\ Q^{-} &= P_1^{+} \cdot P_2^{-} + P_1^{-} \cdot P_2^{+} \end{aligned}$$

Obviously, $P_1 \cdot P_2 = Q^{+} + Q^{-}$. But: Q^{+} and Q^{-} have only positive and negative coefficients, respectively. This entails:

No need for TM Tallying Variable! Just compute each coefficient, and account for total error afterwards based on known max number of contributions.

High Precision Taylor Models - Storage

High precision coefficients are stored as "unevaluated sums of floating point numbers". Let ε be approximate machine epsilon.

Write each high precision coefficient as

$$a = a_0 + a_1 \cdot \varepsilon + a_2 \cdot \varepsilon^2$$

Then each of the a_i has similar magnitude.

AND: Introducing one more variable in the polynomial for "powers of ε" we can utilize completely normal TM polynomial framework.

High Precision Taylor Models - Multiplication

Split each polynomial into two parts:

1. Those coefficients less than ε^{-1} away from cutoff or accumulated remainder bound
2. Those more away (the "higher precision terms")

Multiply the first polynomials in usual way.
Second polynomials: Pre-split each coefficient into two "half length" double precision variables as in "two product" algorithm.
Advantage:

1. Such coefficients can be multiplied without any roundoff error.
2. The pre-splitting cost is linear in length of "higher precision term" polynomials, NOT quadratic