sl::Fad<A,O>

— Version sl-0.1.4 / Autodiff-3.03 —

Forward Automatic Differentiation
for floating point of type A *at order* O

Pierre Aubert paubert@mathinsa.insa-lyon.fr
Nicolas Di Césaré nicolas.dicesare@ann.jussieu.fr

Contents

1 Introduction and Usage 4
3 RM: sl::Fad 5
4 RM: sl::TinyFad 6
2 RM: logical and IO functions 7
2.3 IO operations with Fad and TinyFad 25
2.4 Compatibility section: FADBAD or ADOL-C 26

5 RM: namespace meta 27
Class Graph 28

Abstract

This module provide an easy way to compute derivatives in programs. It provides automatic forward and backward differentiations.

A introduction on automatic differentiation is given in the first chapter of this manual (in french): this is a part Of Nicolas thesis.

This library is under the GPL and you can find it as part of the SL library at:

http://home.cern.ch/ ldeniau/html/sl++.html
or
http://pcmath03.insa-lyon.fr/ paubert

1 Introduction and Usage

Include: /home-local/paubert/logiciels/sl/development/sl++/doc/autodiff1.dxx

3

RM: sl::Fad

4

RM: sl::TinyFad

2 RM: logical and IO functions

Names

2.1 *logical operations of Fad<A, O>* 7

2.2 *logical operations of TinyFad<A, N, O>* 23

2.3 **IO operations with Fad and TinyFad** 25

2.4 **Compatibility section: FADBAD or ADOL-C** 26

2.1

logical operations of Fad<A,O>

Names

2.1.1 **Documentation** 10

2.1.2 **Required typedef(s)** 10

2.1.3 **constructor(s), destructor** 10

2.1.4 **required typedefs** 11

2.1.5 **constructor(s), destructor** 11

2.1.6 **members(s)** 11

2.1.7 **members(s) compatibility with fadbad** 12

2.1.8 **unary operation** 12

2.1.9 **overloaded computed assignement with constant** 13

2.1.10 **overloaded computed assignement with this_t** 13

2.1.11 **overloaded computed assignement with meta::Expr<>** 14

2.1.12 **fast get i-th derivative** 22

2.1.13 **private part** 23

2.1.14 **Explanation** 14

2.1.15 **Example(s)** 14

2.1.16 **Required typedefs** 15
2.1.17 **constructor(s), destructor** 15
2.1.18 **members(s)** 15
2.1.19 **members(s) compatibility with fadbad** 16
2.1.20 **overloaded computed assignement with constant** 16
2.1.21 **overloaded computed assignement with this_t** 17
2.1.22 **overloaded computed assignement with meta::Expr<>** 17
2.1.23 **fast get derivative** 22
2.1.24 **private part** 23

A	**value_t**	*define typedef value_t*
A	**expr_t**	*define typedef expr_t*
		status =0, 1, 2 constant, fad, expression
	this_t	*this_t typedef*

2.1.25 **Explanation** 18
2.1.26 **Example** 18
2.1.27 **constructor(s), destructor** 18
2.1.28 **members(s)** 19
2.1.29 **members(s) compatibility with fadbad** 19
2.1.30 **unary operation** 20
2.1.31 **overloaded computed assignement with constant** 20
2.1.32 **overloaded computed assignement with this_t** 20
2.1.33 **overloaded computed assignement with meta::Expr<>** 21
2.1.34 **fast get i-th derivative (don't use it), should be private** 21

value_t	**my_v**	*value*
value_t	**my_d** [N]	*derivative (physical allocated zone)*
void	**copy** (value_t const* a)	*copy*
void	**zero** ( )	*zero*
bool	**operator==** (Fad<A, O> const& a, Fad<A, O> const& b)	*operator ==*
bool	**operator==** (Fad<A, O> const& a, A const& b)	

operator ==

bool **operator==** (A const& a, Fad<A, O> const& b)
operator ==

bool **operator!=** (Fad<A, O> const& a, Fad<A, O> const& b)
operator ≠

bool **operator!=** (Fad<A, O> const& a, A const& b)
operator ≠

bool **operator!=** (A const& a, Fad<A, O> const& b)
operator ≠

bool **operator<** (Fad<A, O> const& a, Fad<A, O> const& b)
operator <

bool **operator<** (Fad<A, O> const& a, A const& b)
operator <

bool **operator<** (A const& a, Fad<A, O> const& b)
operator <

bool **operator>** (Fad<A, O> const& a, Fad<A, O> const& b)
operator >

bool **operator>** (Fad<A, O> const& a, A const& b)
operator >

bool **operator>** (A const& a, Fad<A, O> const& b)
operator >

bool **operator<=** (Fad<A, O> const& a, Fad<A, O> const& b)
operator < =

bool **operator<=** (Fad<A, O> const& a, A const& b)
operator < =

bool **operator<=** (A const& a, Fad<A, O> const& b)
operator < =

bool **operator>=** (Fad<A, O> const& a, Fad<A, O> const& b)
operator > =

bool **operator>=** (Fad<A, O> const& a, A const& b)
operator > =

bool **operator>=** (A const& a, Fad<A, O> const& b)
operator > =

bool **operator!** (Fad<A, O> const& a)
operator ! (not)

2.1.1 Documentation

class `Fad<A>` stand for Forward Automatic Differentiation

With this class it is possible to compute the gradient of a function without writing it. A special class has been designed in order to provide this functionality.

Examples are provided below: see `sl/tests/debug/fadiff.cc` or `sl/tests/validate/fadiff*.cc` or `sl/tests/benchmark/fadiff*.cc`

2.1.2 Required typedef(s)

Names

A	**value_t**	*define typedef value_t*
A	**expr_t**	*define typedef expr_t*
value_t*	**derivative_t**	*type of the derivative*
	this_t	*this_t typedef*

2.1.3 constructor(s), destructor

Names

Fad (value_t const v=value_t(0))
default constructor

Fad (unsigned short const sz, value_t const v=value_t(0))
constructor with size and value

Fad (this_t const& rhs)
copy constructor

~Fad () *destructor*

2.1.4 required typedefs

Names

A	**value_t**	*define typedef value_t*
A	**expr_t**	*define typedef expr_t*
value_t*	**derivative_t**	*type of the derivative*
	this_t	*this_t typedef*

2.1.5 constructor(s), destructor

Names

Fad (value_t const v=value_t(0))	*default constructor*
Fad (unsigned short const sz, value_t const v=value_t(0))	*constructor with size and value*
Fad (this_t const& rhs)	*copy constructor*
Fad (meta::Expr<T_expr> const& e)	*copy constructor*
~Fad ()	*destructor*

2.1.6 members(s)

Names

short	**size** () const	*return number of derivatives (is is always lower or equal than nb()*
const&	**operator()** () const	*needed for meta::Expr*
value_t	**getV** () const	*get value*

value_t	**getD** (const unsigned short i) const	*get i-th derivative*
value_t	**getDI** () const	*templated get I-th derivative*
derivative_t	**getD** () const	*get a const poshorter to the array of derivatives*
value_t	**fgetD** (const unsigned short i) const	*get i-th derivative*

2.1.7 members(s) compatibility with fadbad

Names

void	**diff** (unsigned short idx, unsigned short n)	*initialized idx-th derivative to 1 other to 0*
void	**touchg** (unsigned short n)	*allocate and initializes all elements to zero.*
value_t	**root** () const	*return value*
value_t	**d** (unsigned short i) const	*return the i-th derivative*

2.1.8 unary operation

Names

wrap_t	**operator+** ()	*operator+*
wrap_t	**operator-** ()	*operator-*

2.1.9 overloaded computed assignement with constant

Names

this_t&	**operator=** (value_t const a)	*operator= constant*
this_t&	**operator++** ()	*Fad ++*
this_t&	**operator–** ()	*Fad −−*
this_t&	**operator+=** (value_t const& x)	*Fad + = constant*
this_t&	**operator-=** (value_t const& x)	*Fad − = constant*
this_t&	**operator*=** (value_t const& x)	*Fad ∗ = constant*
this_t&	**operator/=** (value_t const& x)	*Fad / = constant*

2.1.10 overloaded computed assignement with this_t

Names

this_t&	**operator=** (this_t const& rhs)	*operator= this_t*
this_t&	**operator+=** (this_t const& x)	*Fad + = Fad (protected against aliasing)*
this_t&	**operator-=** (const this_t & x)	*Fad − = Fad (protected against aliasing)*
this_t&	**operator*=** (const this_t & x)	*Fad ∗ = Fad (protected against aliasing)*
this_t&	**operator/=** (const this_t & x)	*Fad / = Fad (protected against aliasing)*

2.1.11 overloaded computed assignement with meta::Expr<>

Names

this_t& **operator=** (meta::Expr<T_expr> const& e)
Fad = meta::Expr<>

this_t& **operator+=** (meta::Expr<T_expr> const& e)
Fad + = meta::Expr<>

this_t& **operator-=** (meta::Expr<T_expr> const& e)
Fad − = meta::Expr<> (NOT protected against aliasing)

this_t& **operator*=** (meta::Expr<T_expr> const& e)
Fad ∗ = meta::Expr<>

this_t& **operator/=** (meta::Expr<T_expr> const& e)
Fad / = meta::Expr<> (NOT protected against aliasing)

2.1.14 Explanation

Fad stand for Forward Automatic Differentiation. With this class it is possible to compute the hessian of a function without writing it. A special class has been designed in order to provide this functionality.

2.1.15 Example(s)

see `sl/tests/debug/fadiff.cc` or `sl/tests/validate/fadiff*.cc` or `sl/tests/benchmark/fadiff*.cc`

2.1.16 Required typedefs

Names

A	**value_t**	*define typedef value_t*
A	**expr_t**	*define typedef expr_t*
this_t	**derivative_t**	*type of the derivative*
value_t*	**hessian_t**	*type of the hessian*
	this_t	*this_t typedef*

2.1.17 constructor(s), destructor

Names

Fad (value_t const v=value_t(0))
default constructor

Fad (unsigned short const sz, value_t const v=value_t(0))
constructor with size and value

Fad (this_t const& rhs)
copy constructor

Fad (meta::Expr<T_expr> const& e)
copy constructor

~Fad () *destructor*

2.1.18 members(s)

Names

const&	**operator()** () const	*needed for meta::Expr*
value_t	**getH** (const unsigned short i, const unsigned int j) const	*get i, j-th second derivatives*
value_t&	**fgetH** (const unsigned short i, const unsigned short j)	

set i, j-th second order derivative

value_t **fgetH** (const unsigned short i, const unsigned short j) const
get i, j-th second order derivative

hessian_t **getH** () const
get a const pointer to the array of second order derivatives

2.1.19

members(s) compatibility with fadbad

Names

void **diff2** (unsigned short i, unsigned short j, unsigned short n)
initialized $\frac{\partial^2 f}{\partial x_i \partial x_j}$ *to* 1 *others to* 0

2.1.20

overloaded computed assignement with constant

Names

this_t& **operator=** (value_t const a)
operator= constant

this_t& **operator++** ()
Fad ++

this_t& **operator–** ()
Fad −−

this_t& **operator+=** (value_t const& x)
Fad + = constant

this_t& **operator-=** (value_t const& x)
Fad − = constant

this_t& **operator*=** (value_t const& x)
Fad ∗ = constant

this_t& **operator/=** (value_t const& x)
Fad / = constant

2.1.21

overloaded computed assignement with this_t

Names

this_t& **operator**= (this_t const& rhs)
operator= this_t

this_t& **operator+=** (this_t const& x)
Fad + = Fad (protected against aliasing)

this_t& **operator-=** (const this_t & x)
Fad − = Fad (protected against aliasing)

this_t& **operator*=** (const this_t & x)
Fad ∗ = Fad (protected against aliasing)

this_t& **operator/=** (const this_t & x)
Fad / = Fad (protected against aliasing)

2.1.22

overloaded computed assignement with meta::Expr<>

Names

this_t& **operator**= (meta::Expr<T_expr> const& e)
Fad = meta::Expr<>

this_t& **operator+=** (meta::Expr<T_expr> const& e)
Fad + = meta::Expr<>

this_t& **operator-=** (meta::Expr<T_expr> const& e)
Fad − = meta::Expr<> (NOT protected against aliasing)

this_t& **operator*=** (meta::Expr<T_expr> const& e)
Fad ∗ = meta::Expr<>

this_t& **operator/=** (meta::Expr<T_expr> const& e)
Fad / = meta::Expr<> (NOT protected against aliasing)

2.1.25

Explanation

TinyFad stand for Forward Automatic Differentiation. With this class it is possible to compute the gradient of a function without writing it. A special class has been designed in order to provide this functionality.

2.1.26

Example

see `sl/tests/debug/tinyfad.cc` or `sl/tests/validate/tinyfad*.cc` or `sl/tests/benchmark/tinyfad*.cc`

2.1.27

constructor(s), destructor

Names

TinyFad (value_t const v=value_t(0))
default constructor

TinyFad (unsigned short const sz,
value_t const v=value_t(0))
constructor with size and value

TinyFad (this_t const& rhs)
copy constructor

TinyFad (meta::Expr<T_expr> const& e)
copy constructor

~TinyFad () *destructor*

2.1.28

members(s)

Names

short	**size** () const	*return number of derivatives (is is always lower or equal than nb()*
short	**nb** () const	*return number of allocated elements*
const&	**operator()** () const	*needed for meta::Expr*
value_t	**getV** () const	*get value*
value_t	**getD** (const unsigned short i) const	*get i-th derivative*
value_t	**getDI** () const	*templated get I-th derivative*
const*	**getD** () const	*get a const poshorter to the array of derivatives*
bool	**isConstant** () const	*return true if the Fad is a new born or a constant*

2.1.29

members(s) compatibility with fadbad

Names

void	**diff** (unsigned short idx, unsigned short n)	*initialized idx-th derivative to 1 other to 0*
void	**touchg** (unsigned short n)	*allocate and initializes all elements to zero.*
value_t	**root** () const	*return value*
value_t	**d** (unsigned short i) const	*return the i-th derivative*

2.1.30 unary operation

Names

wrap_t	**operator+** ()	*operator+*
wrap_t	**operator-** ()	*operator-*

2.1.31 overloaded computed assignement with constant

Names

this_t&	**operator=** (value_t const a)	*operator= constant*
this_t&	**operator++** ()	*TinyFad ++*
this_t&	**operator–** ()	*TinyFad −−*
this_t&	**operator+=** (value_t const& x)	*TinyFad + = constant*
this_t&	**operator-=** (value_t const& x)	*TinyFad − = constant*
this_t&	**operator*=** (value_t const& x)	*TinyFad ∗ = constant*
this_t&	**operator/=** (value_t const& x)	*TinyFad / = constant*

2.1.32 overloaded computed assignement with this_t

Names

this_t&	**operator=** (this_t const& rhs)	*operator= this_t*
this_t&	**operator+=** (this_t const& x)	

TinyFad + = TinyFad (protected against aliasing)

this_t& **operator-=** (const this_t & x)
TinyFad − = TinyFad (protected against aliasing)

this_t& **operator*=** (const this_t & x)
TinyFad ∗ = TinyFad (protected against aliasing)

this_t& **operator/=** (const this_t & x)
TinyFad / = TinyFad (protected against aliasing)

2.1.33

overloaded computed assignement with meta::Expr<>

Names

this_t& **operator=** (meta::Expr<T_expr> const& e)
TinyFad = meta::Expr<>

this_t& **operator+=** (meta::Expr<T_expr> const& e)
TinyFad + = meta::Expr<>

this_t& **operator-=** (meta::Expr<T_expr> const& e)
TinyFad − = meta::Expr<> (NOT protected against aliasing)

this_t& **operator*=** (meta::Expr<T_expr> const& e)
TinyFad ∗ = meta::Expr<>

this_t& **operator/=** (meta::Expr<T_expr> const& e)
TinyFad / = meta::Expr<> (NOT protected against aliasing)

2.1.34

fast get i-th derivative (don't use it), should be private

Names

value_t **fgetD** (const unsigned short i) const

get i-th derivative

2.1.12

fast get i-th derivative

Names

short	**nb** () const	*return number of allocated elements*
bool	**isOrder0** () const	*return true if the Fad is a new born or a constant*

2.1.23

fast get derivative

Names

short	**nb** () const	*return number of allocated elements in my_h*
bool	**isOrder0** () const	*return true if the Fad is a new born or a constant*
bool	**isOrder1** () const	*return true if the Fad is a first degree polynome*
value_t&	**fgetH** (const unsigned short i)	*fast set i-th second order derivative (not $H(i,\ j)$)*
value_t	**fgetH** (const unsigned short i) const	*fast get i-th second order derivative (not $H(i,\ j)$)*
int	**map** (const unsigned short i, const unsigned short j) const	*map($i,\ j$) map into my_h memory*

2.1.13

private part

Names

short	**my_nb**	*numbre of allocated cells*
short	**my_sz**	*number of derivative*
A	**my_v**	*value*
slrestrict	**my_d**	*derivative (physical allocated zone)*
void	**copy** (A const* a)	*copy*
void	**zero** ( )	*zero*
void	**resize** (unsigned short const sz)	*resize to size sz and fill with value a*

2.1.24

private part

Names

int	**my_nb2**	*nb is the number of physically allocated cells in the hessian*
slrestrict	**my_h**	*hessian (physical allocated zone)*
short	**size2** () const	*return number of composant in my_h*
void	**copy** (A const* a)	*copy*
void	**zero** ( )	*zero*
void	**resize** (unsigned int const sz1)	*resize to size sz*

2.2

logical operations of TinyFad<A,N,O>

Names

bool **operator==** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)
operator ==

bool **operator==** (TinyFad<A, N, O> const& a, A const& b)
operator ==

bool **operator==** (A const& a, TinyFad<A, N, O> const& b)
operator ==

bool **operator!=** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)
operator ≠

bool **operator!=** (TinyFad<A, N, O> const& a, A const& b)
operator ≠

bool **operator!=** (A const& a, TinyFad<A, N, O> const& b)
operator ≠

bool **operator<** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)
operator <

bool **operator<** (TinyFad<A, N, O> const& a, A const& b)
operator <

bool **operator<** (A const& a, TinyFad<A, N, O> const& b)
operator <

bool **operator>** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)
operator >

bool **operator>** (TinyFad<A, N, O> const& a, A const& b)
operator >

bool **operator>** (A const& a, TinyFad<A, N, O> const& b)
operator >

bool **operator<=** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)
operator ≤

bool **operator<=** (TinyFad<A, N, O> const& a, A const& b)
operator ≤

bool **operator<=** (A const& a, TinyFad<A, N, O> const& b)
operator ≤

bool **operator>=** (TinyFad<A, N, O> const& a, TinyFad<A, N, O> const& b)

operator ≥

bool **operator>=** (TinyFad<A, N, O> const& a, A const& b)

operator ≥

bool **operator>=** (A const& a, TinyFad<A, N, O> const& b)

operator ≥

bool **operator!** (TinyFad<A, N, O> const& a)

operator !

2.3

IO operations with Fad and TinyFad

Names

2.3.1 std::ostream&
operator << (std::ostream& os, TinyFad<A, N, O> const& o)
Overloading of standard operator<< for TinyFad<A, N, O> 25

2.3.2 std::ostream&
operator << (std::ostream& os, Fad<A, 1> const& o)
Overloading of standard operator<< for Fad<A, 1> 26

2.3.3 std::ostream&
operator << (std::ostream& os, Fad<A, 2> const& o)
Overloading of standard operator<< for Fad<A, 2> 26

2.3.1

std::ostream& **operator** << (std::ostream& os, TinyFad<A, N, O> const& o)

Overloading of standard operator<< for TinyFad<A,N,O>

Overloading of standard operator<< for TinyFad<A,N,O>

2.3.2

std::ostream& **operator** << (std::ostream& os, Fad<A, 1> const& o
)

Overloading of standard operator<< for Fad<A,1>

Overloading of standard operator<< for Fad<A,1>

2.3.3

std::ostream& **operator** << (std::ostream& os, Fad<A, 2> const& o
)

Overloading of standard operator<< for Fad<A,2>

Overloading of standard operator<< for Fad<A,2>

2.4

Compatibility section: FADBAD or ADOL-C

Names

Fdouble	*first order FAD with double*
FFdouble	*second order FAD with double*
Ffloat	*first order FAD with float*
FFfloat	*second order FAD with float*

5

RM: namespace meta

Class Graph