# MadDiff

Sungho Shin

May 30, 2022

# Contents

# Part I

# Introduction

# Chapter 1

# Introduction

Welcome to the documentation of MadDiff.jl

**Note**

This documentation page is under construction.

**Note**

This documentation is also available in PDF format.

## Chapter 2

# What is MadDiff?

MadDiff.jl is a simple algebraic modeling/differentiation package. MadDiff.jl constructs first and second derivative functions off-line (i.e., prior to calling the optimization solver) by applying operator overloading-based automatic differentiation on functions. The exact derivative functions can be obtained as results.

## Chapter 3

# Bug reports and support

Please report issues and feature requests via the Github issue tracker.

# Part II

# Quick Start

### Nonlinear Expressions

MadDiff.jl provides a flexible user-interface for writing nonlinear expressions and evaluating the expressions and functions. For example,

```
using MadDiff

x = Variable()
p = Parameter()
expr = x[1]^2 + exp(x[2]^p[1])/2 + log(x[3]+p[2])
println(expr) # x[1]^2 + exp(x[2]^p[1])/2 + log(x[3] + p[2])

x0 = [0.,0.5,1.5]
p0 = [2,0.5]

f = function_evaluator(expr)
println("f(x0,p0) = $(f(x0,p0))") # f(x0,p0) = 1.3351598889038159

y0 = zeros(3)
g = gradient_evaluator(expr)
g(y0,x0,p0)
println("g(x0,p0) = $y0") # g(x0,p0) = [0.0, 0.6420127083438707, 0.5]
```

### Nonlinear Programming

MadDiff.jl provides a simple user-interface for creating nonlinear prgogramming models and allows solving the created models using the solvers with `NLPModels.jl` interface (such as NLPModelsIpopt.jl and MadNLP.jl). The syntax is as follows:

```
using MadDiff, NLPModelsIpopt

m = MadDiffModel(; print_level=3)

x = [variable(m; start=mod(i,2)==1 ? -1.2 : 1.) for i=1:1000]
objective(m, sum(100(x[i-1]^2-x[i])^2+(x[i-1]-1)^2 for i=2:1000))
for i=1:998
    constraint(m,
    ↪  3x[i+1]^3+2*x[i+2]-5+sin(x[i+1]-x[i+2])sin(x[i+1]+x[i+2])+4x[i+1]-x[i]exp(x[i]-x[i+1])-3 ==
    ↪  0)
end

instantiate!(m) # this makes the model ready to be solved
ipopt(m)
```

### Use with JuMP

MadDiff.jl can be used as an automatic differentiation backend. The syntax is as follows:

```
using MadDiff, JuMP, Ipopt

m = JuMP.Model(Ipopt.Optimizer)

@variable(m, x[i=1:1000], start=mod(i,2)==1 ? -1.2 : 1.)
@NLobjective(m, Min, sum(100(x[i-1]^2-x[i])^2+(x[i-1]-1)^2 for i=2:1000))
```

```
@NLconstraint(m, [i=1:998],
↪  3x[i+1]^3+2*x[i+2]-5+sin(x[i+1]-x[i+2])sin(x[i+1]+x[i+2])+4x[i+1]-x[i]exp(x[i]-x[i+1])-3 == 0)

optimize!(m; differentiation_backend = MadDiffAD())
```

## Part III

# How it works?

# Part IV

# API Manual

# Chapter 4

# MadDiffCore

## 4.1 MadDiffCore

MadDiffCore.MadDiffCore – Module.

```
MadDiffCore
```

Core algorithm for MadDiff.

source

MadDiffCore.AbstractExpression – Type.

```
AbstractExpression{T <: AbstractFloat}
```

Abstract type for expression, gradient, hessian, entry, and field evaluators.

source

MadDiffCore.Constant – Type.

```
Constant{T <: AbstractFloat} <: Expression{T}
```

Expression for constants.

```
Constant(x::T) where T <: AbstractFloat
```

Returns a Constant with value x.

**Example**

```
julia> e = Constant(1.)
1.0
julia> non_caching_eval(e, [1.,2.,3.])
1.0
```

source

MadDiffCore.Entry – Type.

```
Entry{T <: AbstractFloat}
```

Abstract type for entry evaluators.

source

MadDiffCore.Expression – Type.

```
Expression{T <: AbstractFloat}
```

Abstract type for expression evaluators.

source

MadDiffCore.Expression1 – Type.

```
Expression1{T <: AbstractFloat, F <: Function ,E <: Expression{T}}  <: Expression{T}
```

Expression for univariate function

source

MadDiffCore.Expression2 – Type.

```
Expression2{T <: AbstractFloat, F <: Function,E1, E2} <: Expression{T}
```

Expression for bivariate function

source

MadDiffCore.ExpressionIfElse – Type.

```
ExpressionIfElse{T,E0 <: Expression{T}, E1, E2} <: Expression{T}
```

Expression for ifelse

source

MadDiffCore.ExpressionSum – Type.

```
ExpressionSum{T <: AbstractFloat, E <: Expression{T}, I} <: Expression{T}
```

Expression for a summation of Expressions

source

MadDiffCore.Field – Type.

```
Field{T <: AbstractFloat}
```

Abstract type for field evaluators.

source

MadDiffCore.Gradient – Type.

```
Gradient{T <: AbstractFloat}
```

Abstract type for gradient evaluators.

source

MadDiffCore.Gradient0 – Type.

```
Gradient0{T <: AbstractFloat} <: Gradient{T}
```

Gradient of Variable.

source

MadDiffCore.GradientNull – Type.

```
GradientNull{T <: AbstractFloat} <: Gradient{T}
```

Gradient of Parameter or Constant.

source

MadDiffCore.Hessian – Type.

```
Hessian{T <: AbstractFloat}
```

Abstract type for hessian evaluators.

source

MadDiffCore.Parameter – Type.

```
Parameter{T <: AbstractFloat} <: Expression{T}
```

Expression for parameters.

source

MadDiffCore.Parameter – Method.

```
Parameter(n::Int)
```

Returns a Parameter{Float64} whose index is n

**Example**

```
julia> e = Parameter(3)
p[3]
julia> non_caching_eval(e, [1.,2.,3.], [4.,5.,6.])
6.0
```

source

MadDiffCore.Parameter – Method.

```
Parameter{T}(n::Int) where T <: AbstractFloat
```

Returns a Parameter{T} whose index is n.

source

MadDiffCore.Variable – Type.

```
Variable{T <: AbstractFloat} <: Expression{T}
```

Expression for variables.

source

MadDiffCore.Variable – Method.

```
Variable(n::Int)
```

Returns a `Variable{Float64}` whose index is `n`

**Example**

```
julia> e = Variable(2)
x[2]
julia> non_caching_eval(e, [1.,2.,3.])
2.0
```

source

`MadDiffCore.Variable` – Method.

```
Variable{T}(n::Int) where T <: AbstractFloat
```

Returns a `Variable{T}` whose index is `n`.

source

# Chapter 5

# MadDiffModels

## 5.1 MadDiffModels

`MadDiffModels.MadDiffModels` – Module.

```
MadDiffModels
```

`MadDiffModels` is a submodule of `MadDiff`. `MadDiffModels` allows modeling nonlinear optimization problem of the following form:

```
minimize:   f(x)
subject to: xl ≤   x  ≤ xu
            gl ≤ g(x) ≤ gu,
```

where:

- `x ∈ R^n` is the decision variable.
- `f : R^n -> R` is the objective function
- `g : R^n -> R^m` is the constraint mapping.

The model is constructed as an `NLPModel` (see https://github.com/JuliaSmoothOptimizers/NLPModels.jl), and can be solved with various NLP solvers such as:

- MadNLP (https://github.com/MadNLP/MadNLP.jl)
- Ipopt (https://github.com/JuliaSmoothOptimizers/NLPModelsIpopt.jl)
- Knitro (https://github.com/JuliaSmoothOptimizers/NLPModelsKnitro.jl)

source

`MadDiffModels.Constraint` – Type.

```
Constraint
```

A constraint index of MadDiffModel.

source

`MadDiffModels.MadDiffModel` – Type.

```
MadDiffModel{T <: Real}
```

A mathematical model of a nonlinaer program.

source

MadDiffModels.MadDiffModel – Method.

```
MadDiffModel()
```

Creates an empty MadDiffModel{Float64}.

**Example** `m = MadDiffModel(linear_solver = "ma27")`

source

MadDiffModels.MadDiffModel – Method.

```
MadDiffModel{T}()
```

Creates an empty MadDiffModel{T}.

**Example** `m = MadDiffModel{Float32}()`

source

MadDiffModels.ModelComponent – Type.

```
ModelComponent
```

A model component (eitehr a variable or a parameter) of MadDiffModel.

source

MadDiffModels.constraint – Method.

```
constraint(m::MadDiffModel, e::MadDiffCore.Expression; lb=0., ub=0.)
```

Adds a constraint to MadDiffModel. **Example** "' m = MadDiffModel()

x = [variable(m) for i=1:3] constraint(m, x[1]^2 + 2*sin(x[2]) - exp(x[3]) >= 0) constraint(m, x[1]^4+ x[2]^4 x[3]^4; lb = 0.1, ub = 1.)

source

MadDiffModels.dual – Method.

```
dual(c::Constraint)
```

Retrun the dual of constraint c.

source

MadDiffModels.instantiate! – Method.

```
instantiate!(m::MadDiffModel; sparse = true)
```

Instantiates the model m. The model must be instantiated before solving.

**Example**

```
using MadDiff, NLPModelsIpopt

m = MadDiffModel()

x = [variable(m) for i=1:3]
objective(m, x[1]^2 + x[2]^2 + sin(x[3]))
constraint(m, 3x[2]^2 <= 1.)

instantiate!(m)
ipopt(m)
```

source

MadDiffModels.lower_bound – Method.

```
lower_bound(c::Constraint)
```

Retrun the lower bound of constraint c.

source

MadDiffModels.lower_bound – Method.

```
lower_bound(x::ModelComponent{V}) where V <: MadDiffCore.Variable
```

Retrun the lower bound of variable x.

source

MadDiffModels.objective – Method.

```
objective(m::MadDiffModel, e::MadDiffCore.Expression
```

Sets the objective function for MadDiffModel. Only minimization is supported. **Example**

```
m = MadDiffModel()

x = [variable(m) for i=1:3]
objective(m, x[1]^2 + x[2] + sin(x[3]))
```

source

MadDiffModels.parameter – Method.

```
parameter(m::MadDiffModel{T}, val)
```

Creates a parameter for MadDiffModel with value val. **Example**

```
m = MadDiffModel()

p = parameter(m, 0.5)
```

source

MadDiffModels.set_lower_bound – Method.

```
set_lower_bound(c::Constraint,val)
```

Set the lower bound of constraint c to val.

source

MadDiffModels.set_lower_bound – Method.

```
set_lower_bound(x::ModelComponent{V},val) where V <: MadDiffCore.Variable
```

Set the lower bound of variable 'x' to val.

source

MadDiffModels.set_upper_bound – Method.

```
set_upper_bound(c::Constraint,val)
```

Set the upper bound of constraint c to val.

source

MadDiffModels.set_upper_bound – Method.

```
set_upper_bound(x::ModelComponent{V},val) where V <: MadDiffCore.Variable
```

Set the upper bound of variable 'x' to val.

source

MadDiffModels.setvalue – Method.

```
setvalue(p::ModelComponent{P},val) where P <: MadDiffCore.Parameter
```

Set the value of parameter 'p' to val.

source

MadDiffModels.setvalue – Method.

```
setvalue(x::ModelComponent{V},val) where V <: MadDiffCore.Variable
```

Set the value of variable 'x' to val.

source

MadDiffModels.upper_bound – Method.

```
upper_bound(c::Constraint)
```

Retrun the upper bound of constraint c.

source

MadDiffModels.upper_bound – Method.

```
upper_bound(x::ModelComponent{V}) where V <: MadDiffCore.Variable
```

Retrun the upper bound of variable x.

source

MadDiffModels.value – Method.

```
value(p::ModelComponent{P}) where P <: MadDiffCore.Parameter
```

Return the value of parameter p.

source

MadDiffModels.value – Method.

```
value(x::ModelComponent{V}) where V <: MadDiffCore.Variable
```

Return the value of variable x.

source

MadDiffModels.variable – Method.

```
variable(m::MadDiffModel{T}; lb=-Inf, ub=Inf, start=0.)
```

Creates a variable for MadDiffModel.

**Example**

```
m = MadDiffModel()

x = variable(m; lb = -1, ub = 1, start = 0.5)
```

source

NLPModels.cons! – Method.

```
NLPModels.cons!(m::MadDiffModel,x::AbstractVector,y::AbstractVector)
```

Evaluate the constraints of m at x and store the result in the vector y.

source

NLPModels.get_ifix – Function.

```
get_ifix(m::MadDiffModel)
```

Return the value ifix from MadDiffModel.

source

NLPModels.get_ifree – Function.

```
get_ifree(m::MadDiffModel)
```

Return the value ifree from MadDiffModel.

source

NLPModels.get_iinf – Function.

```
get_iinf(m::MadDiffModel)
```

Return the value iinf from MadDiffModel.

source

NLPModels.get_ilow – Function.

```
get_ilow(m::MadDiffModel)
```

Return the value ilow from MadDiffModel.

source

NLPModels.get_irng – Function.

```
get_irng(m::MadDiffModel)
```

Return the value irng from MadDiffModel.

source

NLPModels.get_islp – Function.

```
get_islp(m::MadDiffModel)
```

Return the value islp from MadDiffModel.

source

NLPModels.get_iupp – Function.

```
get_iupp(m::MadDiffModel)
```

Return the value iupp from MadDiffModel.

source

NLPModels.get_jfix – Function.

```
get_jfix(m::MadDiffModel)
```

Return the value jfix from MadDiffModel.

source

NLPModels.get_jfree – Function.

```
get_jfree(m::MadDiffModel)
```

Return the value jfree from MadDiffModel.

source

NLPModels.get_jinf – Function.

```
get_jinf(m::MadDiffModel)
```

Return the value jinf from MadDiffModel.

source

NLPModels.get_jlow – Function.

```
get_jlow(m::MadDiffModel)
```

Return the value jlow from MadDiffModel.

source

NLPModels.get_jrng – Function.

```
get_jrng(m::MadDiffModel)
```

Return the value jrng from MadDiffModel.

source

NLPModels.get_jupp – Function.

```
get_jupp(m::MadDiffModel)
```

Return the value jupp from MadDiffModel.

source

NLPModels.get_lcon – Function.

```
get_lcon(m::MadDiffModel)
```

Return the value lcon from MadDiffModel.

source

NLPModels.get_lin – Function.

```
get_lin(m::MadDiffModel)
```

Return the value lin from MadDiffModel.

source

NLPModels.get_lin_nnzj – Function.

```
get_lin_nnzj(m::MadDiffModel)
```

Return the value lin_nnzj from MadDiffModel.

source

NLPModels.get_lvar – Function.

```
get_lvar(m::MadDiffModel)
```

Return the value lvar from MadDiffModel.

source

NLPModels.get_minimize – Function.

```
get_minimize(m::MadDiffModel)
```

Return the value minimize from MadDiffModel.

source

NLPModels.get_name – Function.

```
get_name(m::MadDiffModel)
```

Return the value name from MadDiffModel.

source

NLPModels.get_ncon – Function.

```
get_ncon(m::MadDiffModel)
```

Return the value ncon from MadDiffModel.

source

NLPModels.get_nlin – Function.

```
get_nlin(m::MadDiffModel)
```

Return the value nlin from MadDiffModel.

source

NLPModels.get_nln – Function.

```
get_nln(m::MadDiffModel)
```

Return the value nln from MadDiffModel.

source

NLPModels.get_nln_nnzj – Function.

```
get_nln_nnzj(m::MadDiffModel)
```

Return the value nln_nnzj from MadDiffModel.

source

NLPModels.get_nlvb – Function.

```
get_nlvb(m::MadDiffModel)
```

Return the value nlvb from MadDiffModel.

source

NLPModels.get_nlvc – Function.

```
get_nlvc(m::MadDiffModel)
```

Return the value nlvc from MadDiffModel.

source

NLPModels.get_nlvo – Function.

```
get_nlvo(m::MadDiffModel)
```

Return the value nlvo from MadDiffModel.

source

NLPModels.get_nnln – Function.

```
get_nnln(m::MadDiffModel)
```

Return the value nnln from MadDiffModel.

source

NLPModels.get_nnzh – Function.

```
get_nnzh(m::MadDiffModel)
```

Return the value nnzh from MadDiffModel.

source

NLPModels.get_nnzj – Function.

```
get_nnzj(m::MadDiffModel)
```

Return the value nnzj from MadDiffModel.

source

NLPModels.get_nnzo – Function.

```
get_nnzo(m::MadDiffModel)
```

Return the value nnzo from MadDiffModel.

source

NLPModels.get_nvar – Function.

```
get_nvar(m::MadDiffModel)
```

Return the value nvar from MadDiffModel.

source

NLPModels.get_ucon – Function.

```
get_ucon(m::MadDiffModel)
```

Return the value ucon from MadDiffModel.

source

NLPModels.get_uvar – Function.

```
get_uvar(m::MadDiffModel)
```

Return the value uvar from MadDiffModel.

source

NLPModels.get_x0 – Function.

```
get_x0(m::MadDiffModel)
```

Return the value x0 from MadDiffModel.

source

NLPModels.get_y0 – Function.

```
get_y0(m::MadDiffModel)
```

Return the value y0 from MadDiffModel.

source

NLPModels.grad! – Method.

```
NLPModels.grad!(m::MadDiffModel,x::AbstractVector,y::AbstractVector)
```

Evaluate the gradient of m at x and store the result in the vector y.

source

NLPModels.hess_coord! – Method.

```
NLPModels.hess_coord!(m::MadDiffModel,x::AbstractVector,lag::AbstractVector,z::AbstractVector;
↪  obj_weight = 1.0)
```

Evaluate the Lagrangian Hessian of m at primal x, dual lag, and objective weight obj_weight and store the result in the vector zin sparse coordinate format.

source

NLPModels.hess_structure! – Method.

```
NLPModels.hess_structure!(m::MadDiffModel,I::AbstractVector{T},J::AbstractVector{T})
```

Evaluate the structure of the Lagrangian Hessian and store the result in I and J in sparse coordinate format.

source

NLPModels.jac_coord! – Method.

```
NLPModels.jac_coord!(m::MadDiffModel,x::AbstractVector,J::AbstractVector)
```

Evaluate the constraints Jacobian of m at x and store the result in the vector J in sparse coordinate format.

source

NLPModels.jac_structure! – Method.

```
NLPModels.jac_structure!(m::MadDiffModel,I::AbstractVector{T},J::AbstractVector{T})
```

Evaluate the structure of the constraints Jacobian and store the result in I and J in sparse coordinate format.

source

NLPModels.obj – Method.

```
NLPModels.obj(m::MadDiffModel,x::AbstractVector)
```

Return the objective value of m at x.

source

# Chapter 6

# MadDiffMOI

## 6.1 MadDiffMOI

MadDiffMOI.MadDiffMOI – Module.

```
MadDiffMOI
```

MadDiffMOI is a submodule of MadDiff. MadDifMOI allows solving nonlinear optimization problems specified by MathOptInterface (https://github.com/jump-dev/JuMP.jl/tree/od/moi-nonlinear).

source

MadDiffCore.Expression – Method.

Expression(ex::MOI.Nonlinear.Expression; subex = nothing)

Create a MadDiff.Expression from MOI.Expression.

source

MadDiffCore.SparseNLPCore – Method.

```
MadDiffCore.SparseNLPCore(nlp_data::MOI.Nonlinear.Model)
```

Create MadDiffCore.SparseNLPCore from MOI.Nonlinear.Model.

source

MadDiffMOI.MadDiffAD – Type.

```
MadDiffAD() <: MOI.Nonlinear.AbstractAutomaticDifferentiation
```

A differentiation backend for MathOptInterface based on MadDiff

source

MadDiffMOI.MadDiffEvaluator – Type.

```
MadDiffEvaluator <: MOI.AbstractNLPEvaluator
```

A type for callbacks for MathOptInterface's nonlinear model.

source

MathOptInterface.NLPBlockData – Method.

```
MOI.NLPBlockData(evaluator::MadDiffEvaluator)
```

Create MOI.NLPBlockData from MadDiffEvaluator

source

MathOptInterface.Nonlinear.Evaluator – Method.

```
MOI.Nonlinear.Evaluator(model::MOI.Nonlinear.Model, ::MadDiffAD, ::Vector{MOI.VariableIndex})
```

Create a `MOI.Nonlinear.Evaluator` from `MOI.Nonlinear.Model` using `MadDiff`'s AD capability.

source

MathOptInterface.eval_constraint – Method.

```
MOI.eval_constraint(evaluator::MadDiffEvaluator, g, x)
```

Evaluate the gradient of `evaluator` at `x` and store the result in the vector `g`.

source

MathOptInterface.eval_constraint_jacobian – Method.

```
MOI.eval_constraint_jacobian(evaluator::MadDiffEvaluator, J, x)
```

Evaluate the constraints Jacobian of `evaluator` at `x` and store the result in the vector `J` in sparse coordinate format.

source

MathOptInterface.eval_hessian_lagrangian – Method.

```
MOI.eval_hessian_lagrangian(evaluator::MadDiffEvaluator, H, x, σ, μ)
```

Evaluate the Lagrangian Hessian of `evaluator` at primal `x`, dual `μ`, and objective weight `σ` and store the result in the vector `H` in sparse coordinate format.

source

MathOptInterface.eval_objective – Method.

```
MOI.eval_objective(evaluator::MadDiffEvaluator, x)
```

Return the objective value of `evaluator` at `x`.

source

MathOptInterface.eval_objective_gradient – Method.

```
MOI.eval_objective_gradient(evaluator::MadDiffEvaluator, g, x)
```

Evaluate the constraints of `evaluator` at `x` and store the result in the vector `g`.

source

MathOptInterface.hessian_lagrangian_structure – Method.

```
MOI.hessian_lagrangian_structure(evaluator::MadDiffEvaluator)
```

Return the structure of the Lagrangian Hessian in Vector{Tuple{Int,Int}} format.

source

MathOptInterface.jacobian_structure – Method.

```
MOI.jacobian_structure(evaluator::MadDiffEvaluator)
```

Return the structure of the constraints Jacobian in Vector{Tuple{Int,Int}} format.

source