

# MadDiff

Sungho Shin

May 30, 2022

# Contents

<b>Contents</b>	<b>ii</b>
<b>I Introduction</b>	<b>1</b>
1 Introduction	2
2 What is MadDiff?	3
3 Bug reports and support	4
<b>II Quick Start</b>	<b>5</b>
<b>III How it works?</b>	<b>8</b>
<b>IV API Manual</b>	<b>9</b>
4 <b>MadDiffCore</b>	<b>10</b>
4.1 MadDiffCore . . . . .	10
5 <b>MadDiffModels</b>	<b>11</b>
5.1 MadDiffModels . . . . .	11
6 <b>MadDiffMOI</b>	<b>20</b>
6.1 MadDiffMOI . . . . .	20

## **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 programming 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**

## 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:

- $x \in \mathbb{R}^n$  is the decision variable.
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is the objective function
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$  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 nonlineaer 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)
```

Return the upper bound of constraint *c*.

[source](#)

`MadDiffModels.upper_bound` – Method.

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

Return 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.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\\_iring](#) – 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\\_nnzh](#) – Function.

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

Return the value nnzh 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](#)

## Chapter 6

# MadDiffMOI

### 6.1 MadDiffMOI

`MadDiffMOI.MadDiffMOI` – Module.

```
| MadDiffMOI
```

MadDiffMOI is a submodule of MadDiff. MadDiffMOI 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](#)

`MadDiffMOI.MadDiffAD` – Type.

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

A differentiation backend for MathOptInterface based on MadDiff

[source](#)

`MadDiffMOI.MadDiffEvaluator` – Type.

```
| MadDiffEvaluator <: MOI.AbstractNLP evaluator
```

A type for callbacks for MathOptInterface's nonlinear model.

[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](#)