SIMDiff.jl

Sungho Shin

July 30, 2023

Contents

Part I

Introduction

Chapter 1

Introduction

Welcome to the documentation of SIMDiff.jl

Warning

This documentation page is under construction.

Note

This documentation is also available in PDF format.

Chapter 2

What is SIMDiff?

SIMDiff.jl implements SIMD abstraction of nonlinear programs and the automatic differentiation of its functions. SIMDiff.jl expresses the functions in the form of iterables over statically typed data. This allows highly efficient derivative computations based on reverse-mode automatic differentiation.

Chapter 3

Bug reports and support

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

Part II

Quick Start

Chapter 4

Getting Started

SIMDiff can create nonlinear prgogramming models and allows solving the created models using NLP solvers (in particular, those that are interfaced with `NLPModels`, such as NLPModelsIpopt. We now use `SIMDiff` to model the following nonlinear program:

$$\begin{aligned}\min_{\{x_i\}_{i=0}^N} & \sum_{i=2}^{N} 100(x_{i-1}^2 - x_i)^2 + (x_{i-1} - 1)^2 \\ \text{s.t.} & 3x_{i+1}^3 + 2x_{i+2} - 5 + \sin(x_{i+1} - x_{i+2})\sin(x_{i+1} + x_{i+2}) + 4x_{i+1} - x_i e^{x_i - x_{i+1}} - 3 = 0\end{aligned}$$

We model the problem with:

```
using SIMDiff
```

We set

```
N = 10000
```

```
10000
```

First, we create a `SIMDiff.Core`.

```
c = SIMDiff.Core()
```

```
SIMDiff.Core{Float64, Vector{Float64}, Nothing}(SIMDiff.ObjectiveNull(), SIMDiff.ConstraintNull(),
     0, 0, 0, 0, 0, 0, 0, 0, Float64[], Float64[], Float64[], Float64[], Float64[], Float64[],
     nothing)
```

The variables can be created as follows:

```
x = SIMDiff.variable(
    c, N;
    start = (mod(i,2)==1 ? -1.2 : 1. for i=1:N)
)
```

```
SIMDiff.Variable{Tuple{Int64}, Int64}((10000,), 0)
```

The objective can be set as follows:

```
SIMDiff.objective(c, 100*(x[i-1]^2-x[i])^2+(x[i-1]-1)^2 for i in 2:N)
```

```
SIMDiff.Objective{SIMDiff.ObjectiveNull, SIMDiff.Func{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof
    (*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(^), SIMDiff.
    Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64},
    SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}}, Int64}}, SIMDiff.Node2{typeof(^),
    SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.
    Par, Int64}, Int64}}, Int64}, Int64}}, SIMDiff.Compressor{Tuple{Int64, Int64, Int64}}, SIMDiff.
    Compressor{NTuple{4, Int64}}}, UnitRange{Int64}}(SIMDiff.ObjectiveNull(), SIMDiff.Func{SIMDiff.
    Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Node2{typeof
    (-), SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-),
    SIMDiff.Par, Int64}, Int64}}, Int64}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
    }}}, Int64}}, SIMDiff.Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, Int64}}, SIMDiff.Compressor{
    Tuple{Int64, Int64, Int64}}, SIMDiff.Compressor{NTuple{4, Int64}}}(SIMDiff.Node2{typeof(+),
    SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Node2{
    typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64},
    Int64}}, Int64}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}}, Int64}}, SIMDiff.
    Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
    typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, Int64}}(SIMDiff.Node2{typeof(*), Int64, SIMDiff
    .Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{
    typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, SIMDiff.Var{SIMDiff.
    Node2{typeof(+), SIMDiff.Par, Int64}}}, Int64}}(100, SIMDiff.Node2{typeof(^), SIMDiff.Node2{
    typeof(-), SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-)
    , SIMDiff.Par, Int64}, Int64}}, Int64}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
    }}}, Int64}(SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, SIMDiff.Var{SIMDiff.Node2{
    typeof(+), SIMDiff.Par, Int64}}}(SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+),
    SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}(SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.
    Node2{typeof(-), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}(
    SIMDiff.Par(), 1), 0)), 2), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}(SIMDiff.
    Node2{typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 0))), 2)), SIMDiff.Node2{typeof(^), SIMDiff.
    Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Par,
    Int64}, Int64}}, Int64}, Int64}(SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+),
    SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}(SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.
    Node2{typeof(-), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}(
    SIMDiff.Par(), 1), 0)), 1), 2)), SIMDiff.Compressor{Tuple{Int64, Int64, Int64}}((1, 2, 1)),
    SIMDiff.Compressor{NTuple{4, Int64}}((1, 2, 3, 1)), 0, 0, 0, 2, 3), 2:10000)
```

The constraints can be set as follows:

```
SIMDiff.constraint(
    c,
    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
    for i in 1:N-2)
```

```
SIMDiff.Constraint{SIMDiff.ConstraintNull, SIMDiff.Func{SIMDiff.Node2{typeof(-), SIMDiff.Node2{
    typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.
    Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.
    Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff.Node2{
    typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
```

```
Int64}, Int64}}}}, Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.Node2{
typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+),
 SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*), Int64,
SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}},
SIMDiff.Node2{typeof(*), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.
Node1{typeof(exp), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}}}}}, Int64}, SIMDiff.Compressor{NTuple{10, Int64}}, SIMDiff.Compressor{NTuple{17, Int64
}}}, UnitRange{Int64}}(SIMDiff.ConstraintNull(), SIMDiff.Func{SIMDiff.Node2{typeof(-), SIMDiff.
Node2{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-),
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff
.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.
Par, Int64}, Int64}}}}, Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.
Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2
{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{
typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*),
Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64
}}}}, SIMDiff.Node2{typeof(*), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}},
SIMDiff.Node1{typeof(exp), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff
.Par, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
}, Int64}}}}}}, Int64}, SIMDiff.Compressor{NTuple{10, Int64}}, SIMDiff.Compressor{NTuple{17,
Int64}}}(SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.
Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*),
Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2
{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, Int64}, SIMDiff.Node2{
typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+),
 SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.
Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+),
SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, SIMDiff.Node2{typeof(*), SIMDiff.Var{
SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Node1{typeof(exp), SIMDiff.Node2{typeof
(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}, Int64}(SIMDiff.Node2{typeof(-),
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff.Node2{typeof(*),
Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64
}}}}, Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(-),
SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}},
SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}},
SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff
.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.
Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, SIMDiff.Node2{
typeof(*), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Node1{typeof(exp),
 SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Var
{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}(SIMDiff.
```

```
Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(+),
SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+),
SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff.Node2{typeof(*), Int64,
SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}},
Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(-), SIMDiff.Var
{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, SIMDiff.Node1{
typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof
(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
 SIMDiff.Par, Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{
typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}(SIMDiff.Node2{typeof(+),
SIMDiff.Node2{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{
typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}, Int64}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.
Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}, Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{
typeof(sin), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof
(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
 SIMDiff.Par, Int64}, Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var
{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}(SIMDiff.Node2
{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^),
SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}},
Int64}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}, Int64}}}}, Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof
(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof
(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.
Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}(SIMDiff.Node2{typeof(*)
, Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
SIMDiff.Par, Int64}, Int64}}, Int64}}(3, SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{
typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}(SIMDiff.Var{SIMDiff.
Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+),
SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
}(SIMDiff.Par(), 1), 0)), 3)), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}(2, SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.
Node2{typeof(+), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}(
SIMDiff.Par(), 2), 0)))), 5), SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.Node2{
typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof
(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+),
 SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}(SIMDiff.Node1{typeof(sin), SIMDiff.
Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}}}(SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}, Int64}}}(SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 1), 0)),
 SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}(
SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 2), 0)))), SIMDiff.Node1{typeof(sin), SIMDiff.
Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
Int64}, Int64}}}}(SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
```

```
    typeof(+), SIMDiff.Par, Int64}, Int64}}}(SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
    typeof(+), SIMDiff.Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+),
    SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 1), 0)),
     SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}(
    SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{
    typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 2), 0)))))), SIMDiff.Node2{typeof(*), Int64,
    SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}(4,
    SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}(
    SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}(SIMDiff.Node2{
    typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 1), 0)))), SIMDiff.Node2{typeof(*), SIMDiff.Var{
    SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Node1{typeof(exp), SIMDiff.Node2{typeof
    (-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}(SIMDiff.Var{SIMDiff.Node2{typeof
    (+), SIMDiff.Par, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 0)),
    SIMDiff.Node1{typeof(exp), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff
    .Par, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
    }, Int64}}}}(SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}},
     SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}(
    SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.Par,
    Int64}(SIMDiff.Par(), 0)), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff
    .Par, Int64}, Int64}}(SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64},
    Int64}(SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}(SIMDiff.Par(), 1), 0))))))), 3), SIMDiff.
    Compressor{NTuple{10, Int64}}((1, 2, 1, 2, 1, 2, 1, 3, 3, 1)), SIMDiff.Compressor{NTuple{17,
    Int64}}((1, 1, 2, 3, 1, 2, 3, 1, 3, 4, 2, 5, 5, 1, 6, 5, 6)), 0, 0, 29997, 3, 6), 1:9998)
```

Finally, we create an NLPModel.

```
m = SIMDiff.Model(c)
```

```
SIMDiff.Model{Float64, Vector{Float64}, Nothing, SIMDiff.Objective{SIMDiff.ObjectiveNull, SIMDiff.
    Func{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.
    Node2{typeof(-), SIMDiff.Node2{typeof(^), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{
    typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.
    Par, Int64}}}, Int64}}, SIMDiff.Node2{typeof(^), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.
    Node2{typeof(+), SIMDiff.Node2{typeof(-), SIMDiff.Par, Int64}, Int64}}, Int64}, Int64}}, SIMDiff
    .Compressor{Tuple{Int64, Int64, Int64}}, SIMDiff.Compressor{NTuple{4, Int64}}}, UnitRange{Int64
    }}, SIMDiff.Constraint{SIMDiff.ConstraintNull, SIMDiff.Func{SIMDiff.Node2{typeof(-), SIMDiff.
    Node2{typeof(-), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(-),
    SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(*), Int64, SIMDiff.Node2{typeof(^), SIMDiff.Var{
    SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, Int64}}, SIMDiff
    .Node2{typeof(*), Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.
    Par, Int64}, Int64}}}}, Int64}, SIMDiff.Node2{typeof(*), SIMDiff.Node1{typeof(sin), SIMDiff.
    Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
    Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par,
    Int64}, Int64}}}}, SIMDiff.Node1{typeof(sin), SIMDiff.Node2{typeof(+), SIMDiff.Var{SIMDiff.Node2
    {typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}, SIMDiff.Var{SIMDiff.Node2{
    typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64}}}}}}, SIMDiff.Node2{typeof(*),
    Int64, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}, Int64
    }}}}, SIMDiff.Node2{typeof(*), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64}},
    SIMDiff.Node1{typeof(exp), SIMDiff.Node2{typeof(-), SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff
    .Par, Int64}}, SIMDiff.Var{SIMDiff.Node2{typeof(+), SIMDiff.Node2{typeof(+), SIMDiff.Par, Int64
    }, Int64}}}}}}, Int64}, SIMDiff.Compressor{NTuple{10, Int64}}, SIMDiff.Compressor{NTuple{17,
    Int64}}}, UnitRange{Int64}}}
  Problem name: Generic
   All variables: ████████████████████ 10000  All constraints: ████████████████████ 9998
```

```
            free: ████████████████████ 10000             free: ····················· 0
           lower: ····················· 0               lower: ····················· 0
           upper: ····················· 0               upper: ····················· 0
         low/upp: ····················· 0             low/upp: ····················· 0
           fixed: ····················· 0               fixed: ████████████████████ 9998
          infeas: ····················· 0              infeas: ····················· 0
            nnzh: ( 99.82% sparsity)   89985           linear: ····················· 0
                                                    nonlinear: ████████████████████ 9998
                                                         nnzj: ( 99.97% sparsity)   29994

SIMDiff.Counters(0, 0, 0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0)
```

To solve the problem with Ipopt,

```
using NLPModelsIpopt
sol = ipopt(m);
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

This is Ipopt version 3.14.13, running with linear solver MUMPS 5.6.0.

Number of nonzeros in equality constraint Jacobian...:    29994
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:    89985

Total number of variables............................:    10000
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:     9998
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  2.5405160e+06 2.48e+01 2.73e+01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.3512419e+06 1.49e+01 8.27e+01  -1.0 2.20e+00    -  1.00e+00 1.00e+00f  1
   2  1.5156131e+05 4.28e+00 1.36e+02  -1.0 1.43e+00    -  1.00e+00 1.00e+00f  1
   3  6.6755024e+01 3.09e-01 2.18e+01  -1.0 5.63e-01    -  1.00e+00 1.00e+00f  1
   4  6.2338933e+00 1.73e-02 8.47e-01  -1.0 2.10e-01    -  1.00e+00 1.00e+00h  1
   5  6.2324586e+00 1.15e-05 8.16e-04  -1.7 3.35e-03    -  1.00e+00 1.00e+00h  1
   6  6.2324586e+00 8.36e-12 7.97e-10  -5.7 2.00e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 6

                                   (scaled)                 (unscaled)
Objective...............:   7.8692659500479645e-01    6.2324586324379885e+00
Dual infeasibility......:   7.9743417331632266e-10    6.3156786526652763e-09
Constraint violation....:   8.3555384833289281e-12    8.3555384833289281e-12
```

```
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e+00
Overall NLP error.......:   7.9743417331632266e-10    6.3156786526652763e-09


Number of objective function evaluations             = 7
Number of objective gradient evaluations             = 7
Number of equality constraint evaluations            = 7
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 7
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 6
Total seconds in IPOPT                               = 1.254

EXIT: Optimal Solution Found.
```

The solution `sol` contains the field `sol.solution` holding the optimized parameters.

Part III

API Manual

Chapter 5

SIMDiff

`SIMDiff.Core` – Type.

SIMDiff.Core

A core data object used for creating `SIMDiff.Model`.

SIMDiff.Core()

Returns `SIMDiff.Core` for creating `SIMDiff.Model{Float64,Vector{Float64}}`

source

`SIMDiff.Core` – Method.

SIMDiff.Core(S::Type)

Returns `SIMDiff.Core` for creating `SIMDiff.Model{T,VT}`, where VT <: S

source

`SIMDiff.Model` – Type.

SIMDiff.Model <: NLPModels.AbstractNLPModel

An NLP model with SIMDiff backend

source

`SIMDiff.Model` – Method.

SIMDiff.Model(core)

source