LogΣexp benchmarks

`@turbo` loops from LoopVectorization.jl are 10x faster than `LogExpFunctions.logsumexp!`. How can this performance improvement be applied to cases where we have `FowardDiff.Dual` types instead of `Float64`s?

```
using LoopVectorization
```

```
using Tullio , LogExpFunctions , ForwardDiff , BenchmarkTools , Test , Pkg
```

```
versioninfo()
```

```
Julia Version 1.8.1
Commit afb6c60d69 (2022-09-06 15:09 UTC)
Platform Info:
  OS: Windows (x86_64-w64-mingw32)
  CPU: 16 × Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-13.0.1 (ORCJIT, skylake-avx512)
  Threads: 8 on 16 virtual cores
Environment:
  JULIA_DEPOT_PATH = D:\libraries\julia
  JULIA_NUM_THREADS = 8
  JULIA_PKGDIR = D:\libraries\julia
  JULIA_PKG_DEVDIR = D:\libraries\julia\dev
  JULIA_REVISE_WORKER_ONLY = 1
```

```
Pkg.status()
```

```
Status `C:\Users\magerton\AppData\Local\Temp\jl_39PDRM\Project.toml`
  [6e4b80f9] BenchmarkTools v1.3.1
  [f6369f11] ForwardDiff v0.10.32
  [2ab3a3ac] LogExpFunctions v0.3.18
  [bdcacae8] LoopVectorization v0.12.131
  [bc48ee85] Tullio v0.3.5
  [44cfe95a] Pkg v1.8.0
  [8dfed614] Test
```

```
[-0.401383, -0.724229, -0.364853]
```

```
begin
    n,k = 1000, 3
    randX = rand(n,k)
    theta0 = randn(k)
end
```

```
[Dual{ForwardDiff.Tag{DataType, Float64}}(-0.401383,1.0,0.0,0.0), Dual{ForwardDiff.Tag{I
```

```
begin
    cfg = ForwardDiff.GradientConfig(Nothing, theta0)
    thetad = cfg.duals
    ForwardDiff.seed!(thetad, theta0, cfg.seeds)
end
```

```
begin
    XF = randX.*theta0'
    XD = randX.*thetad'

    XFtmp = similar(XF)
    XDtmp = similar(XD)

    VbarF = Vector{eltype(XF)}(undef, n)
    VbarD = Vector{eltype(XD)}(undef, n)

    tmp_maxF = similar(VbarF)
    tmp_maxD = similar(VbarD)

    tmp_cart = Vector{CartesianIndex{2}}(undef, n)
end;
```

logsumexp_simd!

using base SIMD loops

```
"using base SIMD loops"
function logsumexp_simd!(Vbar, tmp_max, X)
    n,k = size(X)
    maximum!(tmp_max, X)
    fill!(Vbar, 0)
    @inbounds for j in 1:k
        @simd for i in 1:n
            Vbar[i] += exp(X[i,j] - tmp_max[i])
        end
    end
    @inbounds @simd for i in 1:n
        Vbar[i] = log(Vbar[i]) + tmp_max[i]
    end
    return Vbar
end

```

logsumexp_vanilla!

vanilla loop with no @simd

```
"vanilla loop with no @simd"
function logsumexp_vanilla!(Vbar, tmp_max, X)
    n,k = size(X)
    maximum!(tmp_max, X)
    fill!(Vbar, 0)
    for i in 1:n, j in 1:k
        Vbar[i] += exp(X[i,j] - tmp_max[i])
    end
    for i in 1:n
        Vbar[i] = log(Vbar[i]) + tmp_max[i]
    end
    return Vbar
end
```

logsumexp_turbo!

using `LoopVectorization.@turbo` loops

NOTE - not compatible with `ForwardDiff.Dual` numbers!

```
"""
using `LoopVectorization.@turbo` loops

**NOTE** - not compatible with `ForwardDiff.Dual` numbers!
"""
function logsumexp_turbo!(Vbar, tmp_max, X)
    n,k = size(X)
    maximum!(tmp_max, X)
    fill!(Vbar, 0)
    @turbo for i in 1:n, j in 1:k
        Vbar[i] += exp(X[i,j] - tmp_max[i])
    end
    @turbo for i in 1:n
        Vbar[i] = log(Vbar[i]) + tmp_max[i]
    end
    return Vbar
end
```

logsumexp_vmap!

using `LoopVectorization` `vmap` convenience fcts

NOTE - this DOES work with `ForwardDiff.Dual` numbers!

```
"""
using `LoopVectorization` `vmap` convenience fcts

**NOTE** - this DOES work with `ForwardDiff.Dual` numbers!
"""
function logsumexp_vmap!(Vbar, tmp_max, X, Xtmp)
    maximum!(tmp_max, X)
    n = size(X,2)
    for j in 1:n
        Xtmpj = view(Xtmp, :, j)
        Xj    = view(X, :, j)
        vmap!((xij, mi) -> exp(xij-mi), Xtmpj, Xj, tmp_max)
    end
    Vbartmp = vreduce(+, Xtmp; dims=2)
    vmap!((vi,mi) -> log(vi) + mi, Vbar, Vbartmp, tmp_max)
    return Vbar
end
```

logsumexp_tullio1

Using tullio

```
"Using tullio"
function logsumexp_tullio1(Vbar, tmp_max, X)
    @tullio avx=true (max) tmp_max[i] = X[i,j]
    @tullio avx=true Vbar[i] = exp(X[i,j] - tmp_max[i])
    @tullio avx=true Vbar[i] = log1p(Vbar[i]-1) + tmp_max[i]
  end
```

```
DefaultTestSet("Check fcts are correct", [DefaultTestSet("Floats", [], 5, false, false,
```

```
@testset "Check fcts are correct" begin
    @testset "Floats" begin
                bmark_F = logsumexp!(         VbarF,           XF)
        @test bmark_F ≈ logsumexp_simd!(   VbarF, tmp_maxF, XF)
        @test bmark_F ≈ logsumexp_vanilla!(VbarF, tmp_maxF, XF)
        @test bmark_F ≈ logsumexp_turbo!(  VbarF, tmp_maxF, XF)
        @test bmark_F ≈ logsumexp_vmap!(   VbarF, tmp_maxF, XF, XFtmp)
        @test bmark_F ≈ logsumexp_tullio1( VbarF, tmp_maxF, XF)
    end

    @testset "Duals" begin
        bmark_D = logsumexp!(         VbarD,           XD)
        @test bmark_D ≈ logsumexp_simd!(   VbarD, tmp_maxD, XD)
        @test bmark_D ≈ logsumexp_vanilla!(VbarD, tmp_maxD, XD)
        @test bmark_D ≈ logsumexp_turbo!(  VbarD, tmp_maxD, XD)
        @test bmark_D ≈ logsumexp_vmap!(   VbarD, tmp_maxD, XD, XDtmp)
        @test bmark_D ≈ logsumexp_tullio1( VbarD, tmp_maxD, XD)
    end
end
```

```
Test Summary:                | Pass  Total  Time
Check fcts are correct       |   10     10  0.0s
# #=J:\projects\ShaleDrillingRevisedModel.jl\r                books-scratch\logsumexp-spee
c tests.jl#==#a087f539-8e1b-49af-89f3-cfaf1390ad 1:13 =#:
` `LoopVectorization.check_args` on your inputs failed; running fallback `@inb
c ounds @fastmath` loop instead.
U Use `warn_check_args=false`, e.g. `@turbo warn_check_args=false ...`, to dis
a able this warning.
```

```
Benchmark(evals=1, seconds=5.0, samples=10000)
```

```
begin
    suite = BenchmarkGroup()
    suite["Float64"] = BenchmarkGroup(["Float64"])
    suite["Dual"]    = BenchmarkGroup(["Dual"])

    suite["Float64"]["LogExpFunctions"] =
            @benchmarkable logsumexp!(        $VbarF,            $XF)
    suite["Float64"]["SIMD Loop"]       =
            @benchmarkable logsumexp_simd!(   $VbarF, $tmp_maxF, $XF)
    suite["Float64"]["Vanilla Loop"]    =
            @benchmarkable logsumexp_vanilla!($VbarF, $tmp_maxF, $XF)
    suite["Float64"]["LoopVec @turbo"]          =
            @benchmarkable logsumexp_turbo!(  $VbarF, $tmp_maxF, $XF)
    suite["Float64"]["LoopVec vmap"]            =
            @benchmarkable logsumexp_vmap!(   $VbarF, $tmp_maxF, $XF, $XFtmp)
    suite["Float64"]["Tullio"]          =
            @benchmarkable logsumexp_tullio1( $VbarF, $tmp_maxF, $XF)

    suite["Dual"]["LogExpFunctions"]    =
            @benchmarkable logsumexp!(        $VbarD,            $XD)
    suite["Dual"]["SIMD Loop"]          =
            @benchmarkable logsumexp_simd!(   $VbarD, $tmp_maxD, $XD)
    suite["Dual"]["Vanilla Loop"]       =
            @benchmarkable logsumexp_vanilla!($VbarD, $tmp_maxD, $XD)
    suite["Dual"]["LoopVec @turbo"]             =
            @benchmarkable logsumexp_turbo!(  $VbarD, $tmp_maxD, $XD)
    suite["Dual"]["LoopVec vmap"]               =
            @benchmarkable logsumexp_vmap!(   $VbarD, $tmp_maxD, $XD, $XDtmp)
    suite["Dual"]["Tullio"]             =
            @benchmarkable logsumexp_tullio1( $VbarD, $tmp_maxD, $XD)
end
```

```
results = 2-element BenchmarkTools.BenchmarkGroup:
            tags: []
            "Float64" => 6-element BenchmarkTools.BenchmarkGroup:
                tags: ["Float64"]
                "Vanilla Loop" => Trial(27.900 μs)
                "LoopVec vmap" => Trial(3.900 μs)
                "Tullio" => Trial(24.900 μs)
                "LogExpFunctions" => Trial(40.400 μs)
                "SIMD Loop" => Trial(24.200 μs)
                "LoopVec @turbo" => Trial(3.100 μs)
            "Dual" => 6-element BenchmarkTools.BenchmarkGroup:
                tags: ["Dual"]
                "Vanilla Loop" => Trial(45.400 μs)
                "LoopVec vmap" => Trial(42.500 μs)
                "Tullio" => Trial(54.000 μs)
                "LogExpFunctions" => Trial(63.400 μs)
                "SIMD Loop" => Trial(38.600 μs)
                "LoopVec @turbo" => Trial(317.700 μs)
```

```
• results = run(suite, verbose=true)
```

```
1 #=╠═╡ J:\projects\ShaleDrillingRevisedModel\notebooks-scratch\logsumexp-spee
  ctests.jl#==#a087f539-8e1b-49af-89f3-cfa15 0a0f1:13 =#:
  `LoopVectorization.check_args` on your inpu s failed; running fallback `@inb
  ounds @fastmath` loop instead.
  Use `warn_check_args=false`, e.g. `@turbo w rn_check_args=false ...`, to dis
  able this warning.
```

```
(1/2) benchmarking "Float64"...
  (1/6) benchmarking "Vanilla Loop"...
  done (took 1.1691578 seconds)
  (2/6) benchmarking "LoopVec vmap"...
  done (took 0.8386068 seconds)
  (3/6) benchmarking "Tullio"...
  done (took 1.0526673 seconds)
  (4/6) benchmarking "LogExpFunctions"...
  done (took 1.2829713 seconds)
  (5/6) benchmarking "SIMD Loop"...
  done (took 1.0408769 seconds)
  (6/6) benchmarking "LoopVec @turbo"...
  done (took 0.7813958 seconds)
done (took 6.9578793 seconds)
(2/2) benchmarking "Dual"...
  (1/6) benchmarking "Vanilla Loop"...
  done (took 1.3210121 seconds)
  (2/6) benchmarking "LoopVec vmap"...
  done (took 1.3511774 seconds)
  (3/6) benchmarking "Tullio"...
  done (took 1.3957374 seconds)
  (4/6) benchmarking "LogExpFunctions"...
  done (took 1.7726994 seconds)
  (5/6) benchmarking "SIMD Loop"...
  done (took 1.206825 seconds)
  (6/6) benchmarking "LoopVec @turbo"...
  done (took 5.7335887 seconds)
done (took 13.4673963 seconds)
```

```
2-element BenchmarkTools.BenchmarkGroup:
  tags: []
  "Float64" => 6-element BenchmarkTools.BenchmarkGroup:
      tags: ["Float64"]
      "Vanilla Loop" => Trial(27.900 μs)
      "LoopVec vmap" => Trial(3.900 μs)
      "Tullio" => Trial(24.900 μs)
      "LogExpFunctions" => Trial(40.400 μs)
      "SIMD Loop" => Trial(24.200 μs)
      "LoopVec @turbo" => Trial(3.100 μs)
  "Dual" => 6-element BenchmarkTools.BenchmarkGroup:
      tags: ["Dual"]
      "Vanilla Loop" => Trial(45.400 μs)
      "LoopVec vmap" => Trial(42.500 μs)
      "Tullio" => Trial(54.000 μs)
      "LogExpFunctions" => Trial(63.400 μs)
      "SIMD Loop" => Trial(38.600 μs)
      "LoopVec @turbo" => Trial(317.700 μs)
```

- `results`

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  3.100 μs … 118.100 μs  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     3.900 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   4.299 μs ±   2.174 μs  ┊ GC (mean ± σ):  0.00% ± 0.00%
```


```
 Memory estimate: 0 bytes, allocs estimate: 0.
```

- `results["Float64"]["LoopVec @turbo"]`

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  24.200 μs … 400.100 μs  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     25.400 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   30.769 μs ±  13.939 μs  ┊ GC (mean ± σ):  0.00% ± 0.00%
```


```
 Memory estimate: 0 bytes, allocs estimate: 0.
```

- `results["Float64"]["SIMD Loop"]`

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  38.600 μs … 501.900 μs  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     38.900 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   45.479 μs ±  15.229 μs  ┊ GC (mean ± σ):  0.00% ± 0.00%
```


```
 Memory estimate: 0 bytes, allocs estimate: 0.
```

- `results["Dual"]["SIMD Loop"]`

]

D
ir
m
el
ir
su
sc
th
w
d

br