```julia
using Pkg; Pkg.activate()   # since we need the extend-DR branch
```

```
Activating project at `~/.julia/environments/v1.10`
```

```julia
using Manifolds, Manopt, ManoptExamples
```

```julia
using NamedColors, Plots, BenchmarkTools
```

```julia
begin
paul_tol = load_paul_tol()
indigo = paul_tol["mutedindigo"]
green = paul_tol["mutedgreen"]
sand = paul_tol["mutedsand"]
olive = paul_tol["mutedolive"]
teal = paul_tol["mutedteal"]
wine = paul_tol["mutedwine"]
grey = paul_tol["mutedgrey"]
end;
```

Numerical Example for the acceleration and inertia on DR

The Rosenbrock problem

```
E = Euclidean(2; field=ℝ)
```

```julia
E = Euclidean(2)
```

```
M = MetricManifold(Euclidean(2; field=ℝ), ManoptExamples.RosenbrockMetric())
```

```julia
M = MetricManifold(E, ManoptExamples.RosenbrockMetric())
```

```julia
a = 2*10^5; b = 1; p0 = [0.1, 0.2]; p_star = [b,b^2];
```

```
f =   RosenbrockCost(200000, 1)
```

```julia
f = ManoptExamples.RosenbrockCost(a,b)
```

```
∇f!! =   RosenbrockGradient!!(200000, 1)
```

```julia
∇f!! = ManoptExamples.RosenbrockGradient!!(a=a, b=b)
```

```
prox_f2 (generic function with 1 method)
```

```julia
begin
# These are described as “being defined” in the paper – I would prefer if that wold be derived.
    # f1, f2 hgere refers to the two summands of Rosenbrock, i.e. f = f1 + f2
    # (a) in-place variants
    function prox_f1!(M, q, λ, p)
        q .= [p[1], (p[2] + 2*a*λ*p[1]^2 ) / (1+2*a*λ) ]
        return q
    end
    function prox_f2!(M, q, λ, p)
        q .= [
            (p[1] + 2*λ*b) / (1+2*λ),
            p[2] - ( 4*λ*(p[1] + 2*λ*b) * (p[1] - b)  + 4 * λ^2 * (p[1]-b)^2 ) / (1+2 * a * λ)^2
        ]
        return q
    end
    prox_f1(M, λ, p) = prox_f1!(M, copy(M, p), λ, p)
    prox_f2(M, λ, p) = prox_f2!(M, copy(M, p), λ, p)
    # The reflections here will work automatically, but I _hate_ phrasings like “can be easilu seen”
end
```

```
sc = StopWhenChangeLess(1.0e-14)
    |Δp| < 1.0e-14: not reached
```

```julia
sc = StopWhenChangeLess(1e-14)
```

Classical Douglas Rachford

```
s1 = # Solver state for `Manopt.jl`s  Douglas Rachford Algorithm
     After 32 iterations

     using an in place reflection.

     ## Stopping Criterion
     |Δp| < 1.0e-14: reached
     This indicates convergence: Yes

     ## Debug
         :Stop = :Stop
         :All = [(:Iteration, "# %-6d"), (:Cost, "f(x): %f"), "\n", 25]

     ## Record
     (Iteration = RecordGroup([RecordIteration(), RecordCost()]),)
```

```julia
# A first simple run
s1 = DouglasRachford(M, f, [prox_f1!, prox_f2!], p0;
    α = i -> 0.5,
    λ = i -> 1.0,
    debug = [:Iteration, :Cost, "\n" , 25, :Stop],
    record = [ :Iteration, :Cost],
    stopping_criterion=sc,
    evaluation = InplaceEvaluation(),
    reflection_evaluation = InplaceEvaluation(),
    return_state=true
)
```

```
Initial f(x): 7220.810000
# 25    f(x): 0.000000
The algorithm performed a step with a change (7.835532217989665e-15) less t
han 1.0e-14.
```

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  65.750 μs …   4.629 ms  ┊ GC (min … max): 0.00% … 96.67%
 Time  (median):     72.041 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   76.446 μs ± 114.521 μs  ┊ GC (mean ± σ):  4.38% ±  2.89%
```

```
  65.8 μs         Histogram: frequency by time         106 μs <

 Memory estimate: 52.59 KiB, allocs estimate: 1757.
```

```julia
# Benchmark
@benchmark DouglasRachford($M, $f, [$prox_f1!, $prox_f2!], $p0;
    α = $(i -> 0.5),
    λ = $(i -> 1.0),
    stopping_criterion = $(sc),
    evaluation = $(InplaceEvaluation()),
    reflection_evaluation = $(InplaceEvaluation()),
)
```

```
q1 =  [1.0, 1.0]
```

```julia
q1 = get_solver_result(s1)
```

```
3.194886863360324e-24
```

```
f(M, q1)
```

Inertia

```
s2 = # Solver state for `Manopt.jl`s  Douglas Rachford Algorithm
     After 23 iterations

     using an in place reflection.

     ## Stopping Criterion
     |Δp| < 1.0e-14: reached
     This indicates convergence: Yes

     ## Debug
         :Stop = :Stop
         :All = [(:Iteration, "# %-6d"), (:Cost, "f(x): %f"), "\n", 10]

     ## Record
     (Iteration = RecordGroup([RecordIteration(), RecordCost()]),)
```

```
s2 = DouglasRachford(M, f, [prox_f1!, prox_f2!], p0;
    α = i -> 0.5,
    λ = i -> 1.0,
    θ = i -> 0.12, # Inertia
    debug = [:Iteration, :Cost, "\n" , 10, :Stop],
    record = [ :Iteration, :Cost],
    stopping_criterion=sc,
    evaluation = InplaceEvaluation(),
    reflection_evaluation = InplaceEvaluation(),
    return_state=true
)
```

```
Initial f(x): 7220.810000
# 10    f(x): 0.000000
# 20    f(x): 0.000000
The algorithm performed a step with a change (3.815356980893987e-15) less t
han 1.0e-14.
```

```
q2 =  [1.0, 1.0]
```

```
q2 = get_solver_result(s2)
```

```
3.549874073494553e-25
```

```
f(M, q2)
```

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  49.875 μs …  4.103 ms  ┊ GC (min … max): 0.00% … 97.14%
 Time  (median):     53.167 μs              ┊ GC (median):    0.00%
 Time  (mean ± σ):   56.115 μs ± 93.080 μs  ┊ GC (mean ± σ):  3.99% ±  2.38%
```

```
  49.9 μs         Histogram: frequency by time        69.5 μs <

 Memory estimate: 38.98 KiB, allocs estimate: 1280.
```

```julia
@benchmark DouglasRachford($M, $f, [$prox_f1!, $prox_f2!], $p0;
    α = $(i -> 0.5),
    λ = $(i -> 1.0),
    θ = $(i -> 0.12),
    stopping_criterion=$sc,
    evaluation = $(InplaceEvaluation()),
    reflection_evaluation = $(InplaceEvaluation()),
)
```

Acceleration

```
s3 = # Solver state for `Manopt.jl`s  Douglas Rachford Algorithm
     After 10 iterations

     using an in place reflection.

     ## Stopping Criterion
     |Δp| < 1.0e-14: reached
     This indicates convergence: Yes

     ## Debug
         :Stop = :Stop
         :All = [(:Iteration, "# %-6d"), (:Cost, "f(x): %f"), "\n", 10]

     ## Record
     (Iteration = RecordGroup([RecordIteration(), RecordCost()]),)
```

```julia
s3 = DouglasRachford(M, f, [prox_f1!, prox_f2!], p0;
    α = i -> 0.5,
    λ = i -> 1.0,
    n = 3,
    debug = [:Iteration, :Cost, "\n" , 10, :Stop],
    record = [ :Iteration, :Cost],
    stopping_criterion=sc,
    evaluation = InplaceEvaluation(),
    reflection_evaluation = InplaceEvaluation(),
    return_state=true
)
```

```
Initial f(x): 7220.810000
# 10    f(x): 0.000000
The algorithm performed a step with a change (2.2232198742534223e-15) less
than 1.0e-14.
```

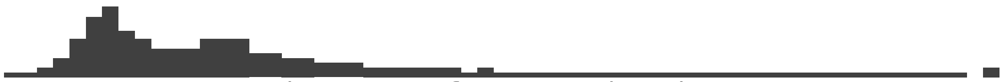
q3 = [1.0, 1.0]

```
q3 = get_solver_result(s3)
```

9.860761315262648e-27

```
f(M, q3)
```

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  63.875 μs …  3.374 ms  ┊ GC (min … max): 0.00% … 96.68%
 Time  (median):     66.292 μs              ┊ GC (median):    0.00%
 Time  (mean ± σ):   69.229 μs ± 82.043 μs  ┊ GC (mean ± σ):  3.08% ±  2.55%
```



```
  63.9 μs         Histogram: frequency by time          78 μs <

 Memory estimate: 44.15 KiB, allocs estimate: 1842.
```

```
@benchmark DouglasRachford($M, $f, [$prox_f1!, $prox_f2!], $p0;
    α = $(i -> 0.5),
    λ = $(i -> 1.0),
    n = 2,
    stopping_criterion=$sc,
    evaluation = $(InplaceEvaluation()),
    reflection_evaluation = $(InplaceEvaluation()),
)
```

Acceleration *and* Interatia

```
s4 = # Solver state for `Manopt.jl`s  Douglas Rachford Algorithm
     After 13 iterations

     using an in place reflection.

     ## Stopping Criterion
     |Δp| < 1.0e-14: reached
     This indicates convergence: Yes

     ## Debug
         :Stop = :Stop
         :All = [(:Iteration, "# %-6d"), (:Cost, "f(x): %f"), "\n", 10]

     ## Record
     (Iteration = RecordGroup([RecordIteration(), RecordCost()]),)
```

```julia
s4 = DouglasRachford(M, f, [prox_f1!, prox_f2!], p0;
    α = i -> 0.5,
    λ = i -> 1.0,
    θ = i -> 0.12, # Inertia
    n = 3,
    debug = [:Iteration, :Cost, "\n" , 10, :Stop],
    record = [ :Iteration, :Cost],
    stopping_criterion=sc,
    evaluation = InplaceEvaluation(),
    reflection_evaluation = InplaceEvaluation(),
    return_state=true
)
```

```
Initial f(x): 7220.810000
# 10    f(x): 0.000000
The algorithm performed a step with a change (8.560244302824867e-15) less t
han 1.0e-14.
```

```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  82.666 μs …   3.397 ms  ┊ GC (min … max): 0.00% … 96.09%
 Time  (median):     84.958 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   88.666 μs ± 92.404 μs   ┊ GC (mean ± σ):  3.07% ±  2.87%
```

```
  82.7 μs         Histogram: frequency by time        99.1 μs <

 Memory estimate: 54.32 KiB, allocs estimate: 2388.
```

```julia
@benchmark DouglasRachford($M, $f, [$prox_f1!, $prox_f2!], $p0;
    α = $(i -> 0.5),
    λ = $(i -> 1.0),
    θ = i -> 0.12, # Inertia
    n = 3,
    stopping_criterion=$sc,
    evaluation = $(InplaceEvaluation()),
    reflection_evaluation = $(InplaceEvaluation()),
)
```

Quasi Newton

A sanity check – quasi Newton should be able to do this reasonably quick as well.

s5 =

```
# Solver state for `Manopt.jl`s Quasi Newton Method
After 213 iterations

## Parameters
* direction update:         limited memory Manopt.InverseBFGS (size 5), projections, an
* retraction method:        ManifoldsBase.ExponentialRetraction()
* vector transport method: ParallelTransport()

## Stepsize
WolfePowellLinesearch(DefaultManifold(), 0.0001, 0.999) with keyword arguments
  * retraction_method = ManifoldsBase.ExponentialRetraction()
  * vector_transport_method = ParallelTransport()

## Stopping Criterion
|Δp| < 1.0e-14: reached
This indicates convergence: Yes

## Debug
    :Stop = :Stop
    :All = [(:Iteration, "# %-6d"), (:Cost, "f(x): %f"), "\n", 100]

## Record
(Iteration = RecordGroup([RecordIteration(), RecordCost()]),)
```

```
s5 = quasi_Newton(E, f, ∇f!!, p0;
    debug = [:Iteration, :Cost, "\n" , 100, :Stop],
    record = [ :Iteration, :Cost],
    memory_size=5,
    stopping_criterion=sc,
    evaluation = InplaceEvaluation(),
    return_state=true
)
```

```
Initial f(x): 7220.810000
# 100    f(x): 0.097674
# 200    f(x): 0.000001
The algorithm performed a step with a change (2.9790409838967277e-15) less
than 1.0e-14.
```

q5 = [1.0, 1.0]

```
q5 = get_solver_result(s5)
```

0.0

```
f(E, q5)
```

```
@benchmark s5 =
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  125.958 μs …   3.608 ms  ┊ GC (min … max): 0.00% … 95.11%
 Time  (median):     131.250 μs               ┊ GC (median):    0.00%
 Time  (mean ± σ):   137.208 μs ± 121.987 μs  ┊ GC (mean ± σ):  3.50% ±  3.78%
```

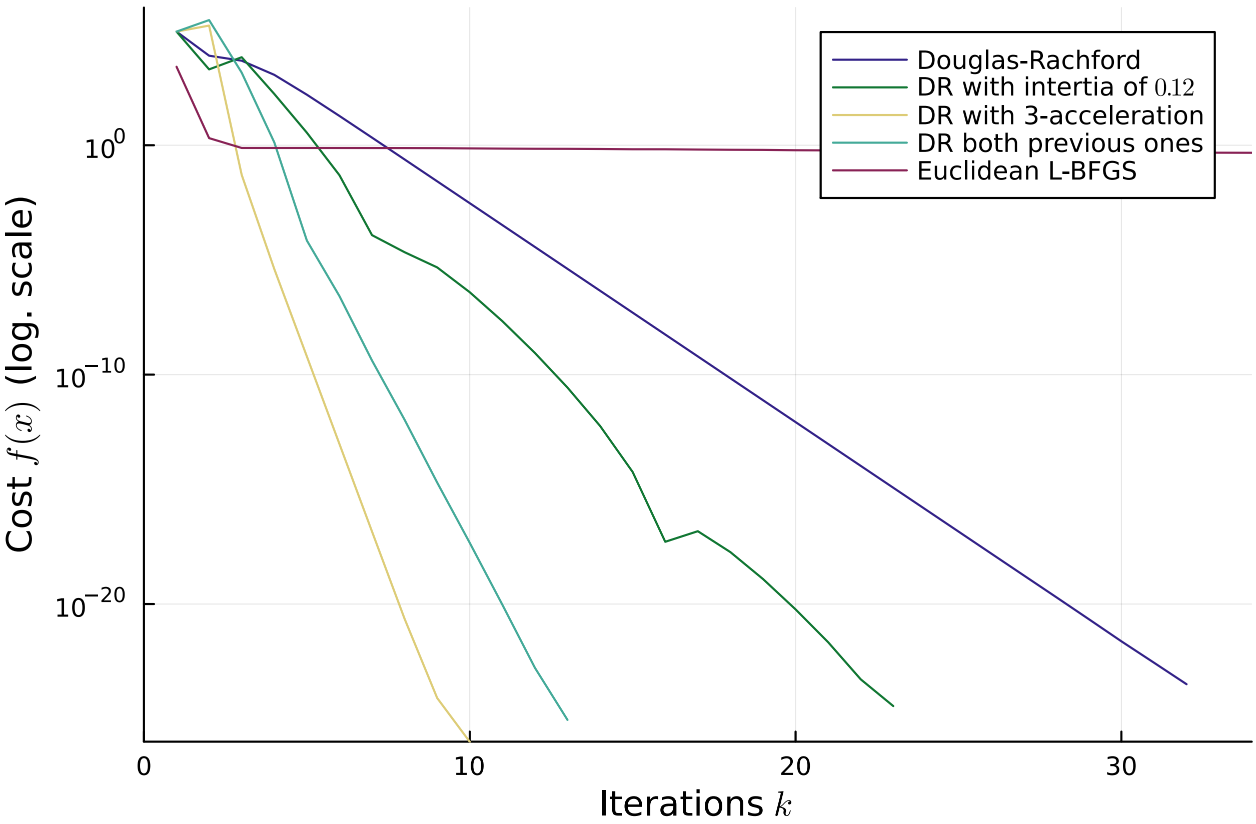
```
  126 μs           Histogram: frequency by time          147 μs <

 Memory estimate: 98.37 KiB, allocs estimate: 1123.
```

```
@benchmark s5 = quasi_Newton($E, $f, $∇f!!, $p0;
    memory_size=5,
    stopping_criterion=$sc,
    evaluation = $(InplaceEvaluation()),
)
```

Summary

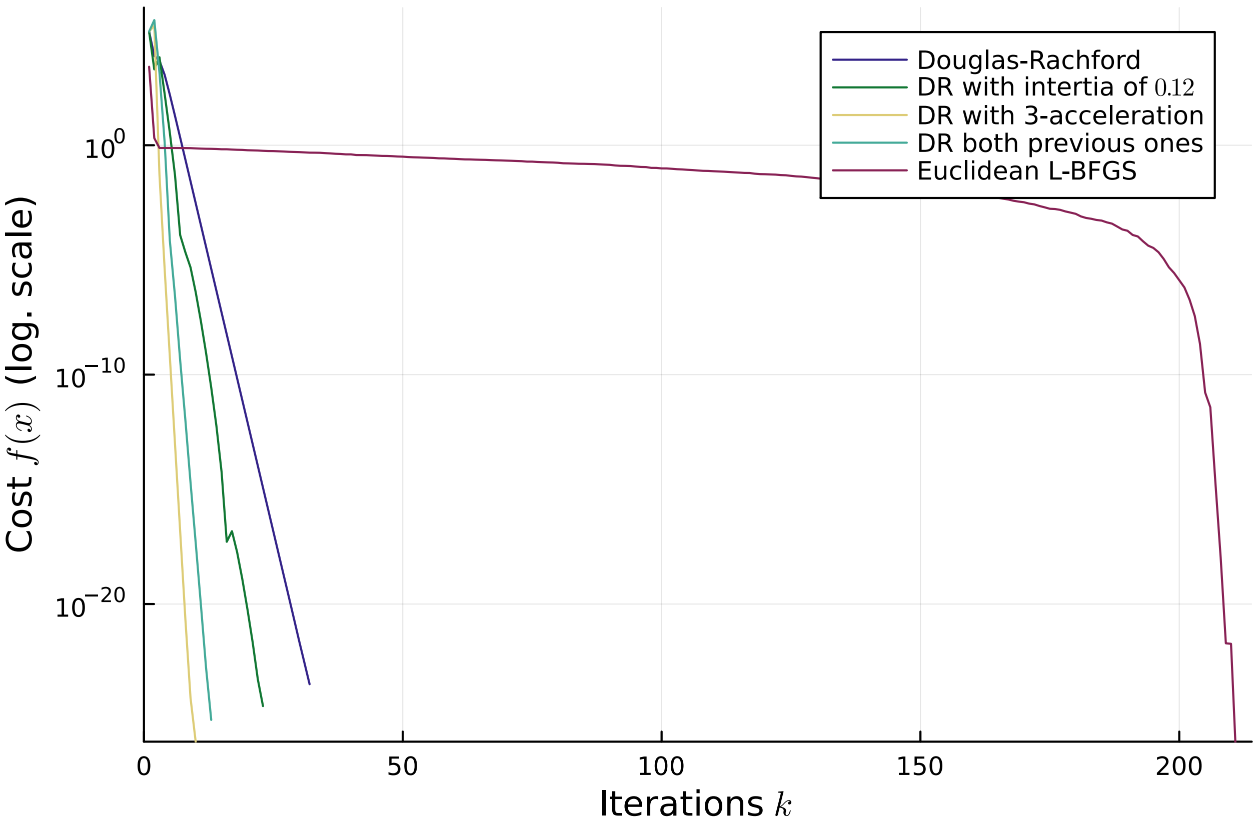
```
iterates = [ [e[1] for e in get_record(s, :Iteration)] for s in [s1,s2, s3, s4, s5]];
```

```
costs = [ [e[2] for e in get_record(s, :Iteration)] for s in [s1,s2, s3, s4, s5]];
```




```julia
begin
    fig = plot(
        xlabel=raw"Iterations $k$", ylabel=raw"Cost $f(x)$ (log. scale)",
        yaxis=:log,
        ylim = (1e-26,1e6),
        xlim = (0,34),
    );
    plot!(fig, iterates[1], costs[1], color=indigo, label="Douglas-Rachford");
    plot!(fig, iterates[2], costs[2], color=green, label=raw"DR with intertia of $0.12$");
    plot!(fig, iterates[3], costs[3], color=sand, label=raw"DR with 3-acceleration");
    plot!(fig, iterates[4], costs[4], color=teal, label=raw"DR both previous ones");
    fig
    plot!(fig, iterates[5], costs[5], color=wine, label=raw"Euclidean L-BFGS");
    fig
end
```

When we look at Quasi Newton, it reaches even numerical zero, but only relatively late:




```
begin
    fig2 = deepcopy(fig)
    plot!(fig2, xlim=(0,214))
end
```