⌘ S

A small note on a certain differential

```
using Manifolds , Manopt , Plots , Random
```

```
MersenneTwister([0x0000002a], DSFMT_state([964434469, 1073036706, 1860149520, 1073503458,
```

```
Random.seed!(42)
```

Let $\mathcal M$ be a manifold, $p,q \in \mathcal M$ and $t \in \mathbb R$ be given.

We consider the function $f\colon T_p\mathcal M \to \mathbb R$

$$f(X) = \frac{1}{2}d_{\mathcal M}^2(q, \exp_p(tX)), \qquad X \in T_p\mathcal M$$

Hence the differential is a map from $T_p\mathcal M$ to $\mathbb R$ since both domain and codomain are their own tangent spaces.

If we use this within minimisation, we can even specify the minmiser $X^* = \frac{1}{t}\log_p q$ aince then the distance is zero.

We decompose f into the functions

$$\begin{aligned} g(s) &= \frac{1}{2}d_{\mathcal M}(q,s), & D_sg(s)[X] &= \langle -\log_s q, X\rangle_s, \quad X \in T_s\mathcal M\\ h(Y) &= \exp_p(Y), & D_Yh(Y)[Z] &= D_Y \exp_p(Y)[Z]\\ i(X) &= tX, & D_Xi(X)[Y] &= tY. \end{aligned}$$

And we use the Chain rule on manifolds, i.e. for f, g the concatenation $f(g(p)) = (f\circ g)(p)$ then the chain rule reads $D(f\circ g)(p)[X] = Df(g(p))\bigl[Dg(p)[X]\bigr]$ (cf. AMS08 p. 195).

For our case this reads $f(X) = g(h(i(X)))$ so that the differential reads

$$Df(X)[Y] = D\,g\big(h(i(X))\big)\Big[Dh(i(X))\big[Di(X)[Y]\big]\Big]$$

where

- $Di(X)[Y] = tY$
- $h(i(X)) = \exp_p(tX)$
- hence $Dh(i(X))\big[Di(X)[Y]\big] = D_e xp_p(tX)[tY]$

so that we finally have with $s = \exp_p tX$ that

$$\begin{aligned} Df(X)[Y] = D\frac{1}{2}d^2_{\mathcal{M}}\big(q, \exp_p(tX)\big)[Y] &= D\frac{1}{2}d^2_{\mathcal{M}}(q, \exp_p tX)\big[D\exp_p(tX)[tY]\big] \\ &= \langle -\log_{\exp_p(tX)} q, D\exp_p(tX)[tY]\rangle_{\exp_p(tX)} \end{aligned}$$

Let's check this in code

```
M = Sphere(2, ℝ)
```

```
M = Sphere(2)
```

```
[-0.140227, 0.832443, -0.536074]
```

```
begin
    sp_p = random_point(M)
    sp_q = random_point(M)
end
```

```
t = 0.3
```

```
t = 0.3
```

```
sp_Xstar = [-3.54386, 4.19783, -3.86796]
```

```
sp_Xstar = 1/t * log(M, sp_p, sp_q)
```

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

```
sp_f(X) = 0.5 * distance(M, sp_q, exp(M, sp_p, t*X)).^2
```

```
0.0
```

```
sp_f(sp_Xstar)
```

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

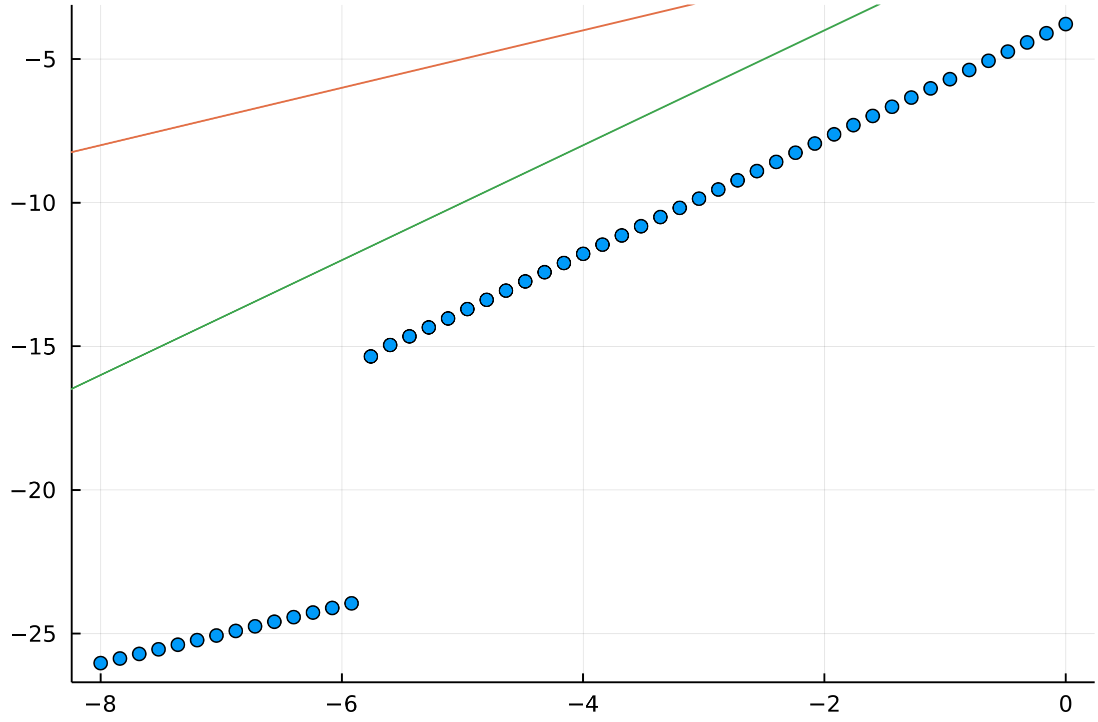
```
sp_Df(X, Y) = inner(M,
    exp(M, sp_p, t*X),
    -log(M, exp(M, sp_p, t*X), sp_q),
    differential_exp_argument(M, sp_p, t*X, t*Y),
)
```

the following ready checks, that in X^* the differential is zero

```
2.9366408416751077e-16
```

```
sp_Df(sp_Xstar, random_tangent(M, sp_p))
```

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




```
check_diff(TangentSpace(M, sp_p), sp_f, sp_Df,
           p = sp_Xstar,  v = random_tangent(M, sp_p))
```

Rotation Manifolds

After demonstrating that the hand-derived differential works well for Spheres, let's check if it also works for Rotation manifolds

We start of by defining some helper function for the ortho-normal basis decomposition of the Jacobi operator for Rotation manifolds.

```
get_basis (generic function with 87 methods)
```

Next, we create the set of points and tangent vectors that we will use for checking our differential.

```
3×3 Array{Float64,2}:
  0.0        3.13874   0.786896
 -3.13874   0.0       -6.44784
 -0.786896  6.44784   0.0
```

```
begin
    R = Rotations(3)
    rot_p = random_point(R)
    rot_q = random_point(R)
    rot_Xstar = 1/t * log(R, rot_p, rot_q)
end
```

And we copy the definitions of the cost functions and the differential (because Pluto doesn't allow the redefinition of exisiting variables).

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

```
rot_f(X) = 0.5 * distance(R, rot_q, exp(R, rot_p, t*X)).^2
```

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

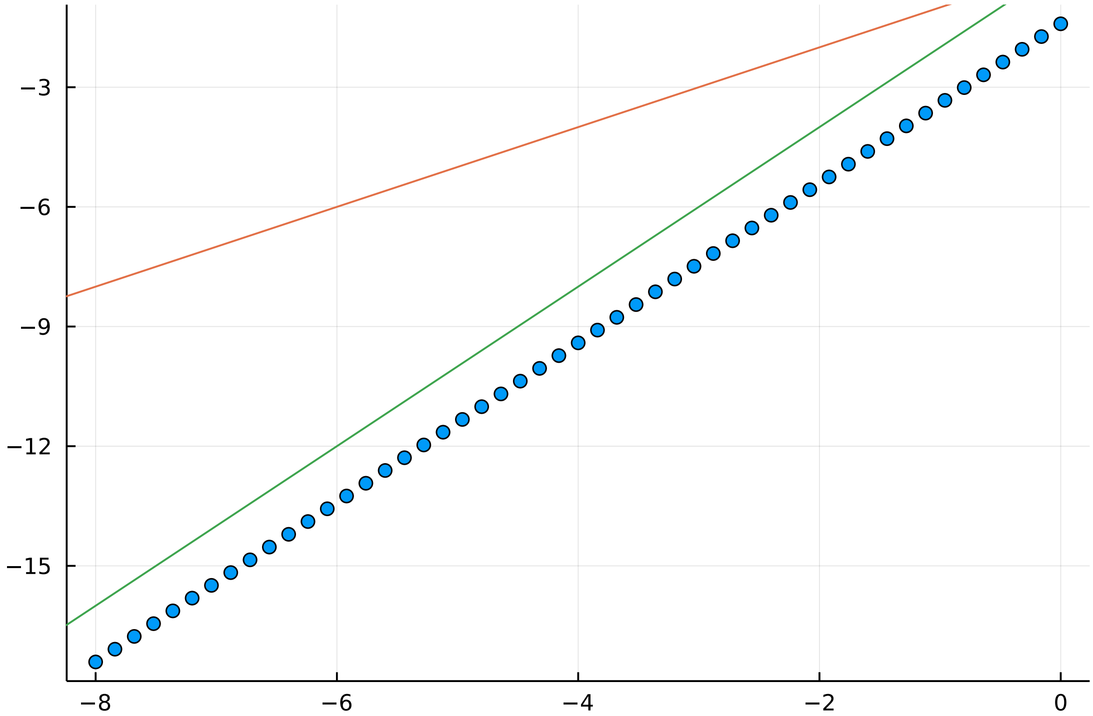
```
rot_Df(X, Y) = inner(R,
    exp(R, rot_p, t*X),
    -log(R, exp(R, rot_p, t*X), rot_q),
    differential_exp_argument(R, rot_p, t*X, t*Y),
)
```

```
2.511412647480955e-31
```

```
rot_f(rot_Xstar)
```

```
4.341530793867162e-18
```

```
rot_Df(rot_Xstar, random_tangent(R, sp_p))
```

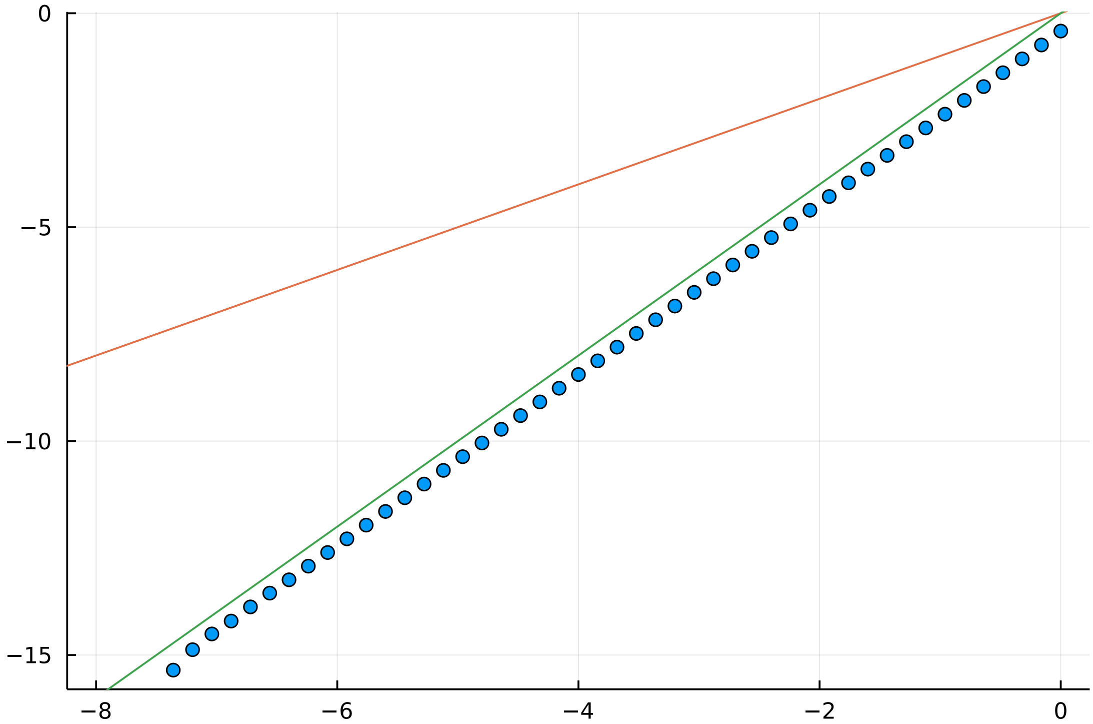



```
check_diff(TangentSpace(R, rot_p), rot_f, rot_Df,
           p = rot_Xstar,  v = random_tangent(R, rot_p))
```

So far the checks look excellent. However, we are currently testing at the X^*, the optimum of f.

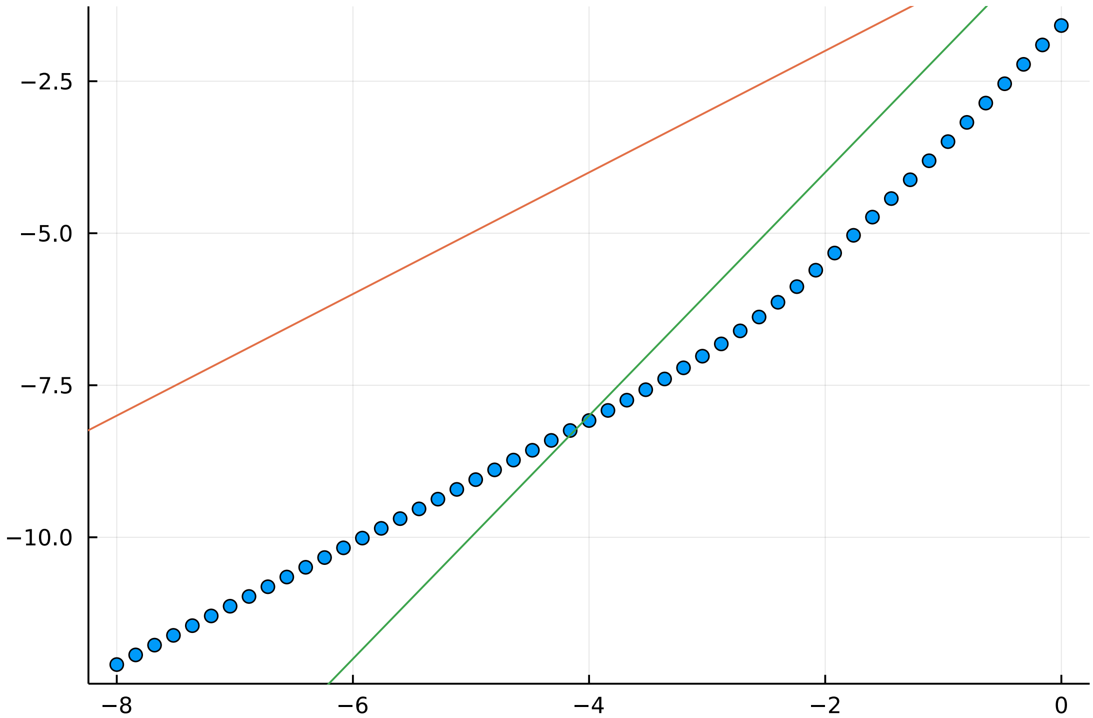
Calling `check_diff()` at random points

While the differential `sp_Df` works for Spheres...




```
check_diff(TangentSpace(M, sp_p), sp_f, sp_Df,
           p = random_tangent(M, sp_p),  v = random_tangent(M, sp_p))
```

... the same code fails for Rotation manifolds.




```
check_diff(TangentSpace(R, rot_p), rot_f, rot_Df,
           p = random_tangent(R, rot_p),  v = random_tangent(R, rot_p))
```

Modifying `βdifferential_exp_arg`

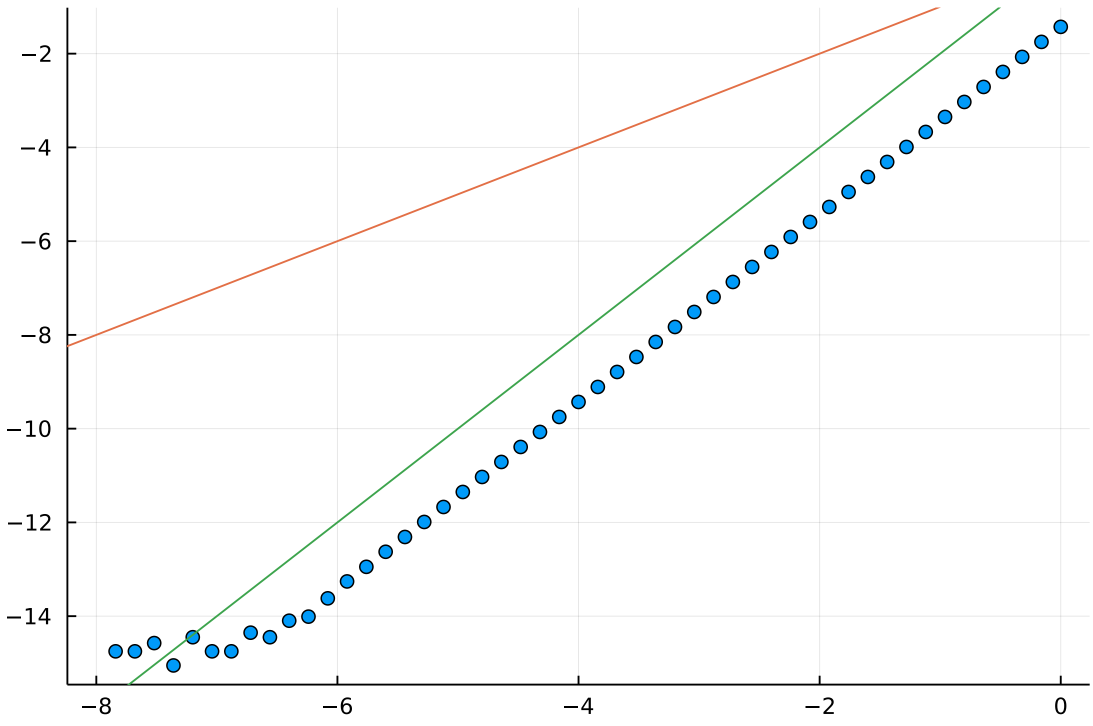
Through more or less random experiments, I found last week that modifying `βdifferential_exp_arg` to take the `t` argument into account, seems to improve the results.

βdifferential_exp_arg_with_time (generic function with 1 method)

```
function βdifferential_exp_arg_with_time(κ, t, d)
    (d == 0 || t == 0) && return 1
    (κ < 0) && return sinh(sqrt(-κ) * t * d) / (t * d * sqrt(-κ))
    (κ > 0) && return sin(sqrt(κ) * t * d) / (t * d * sqrt(κ))
    return 1 # curvature zero
end
```

rot_Df2 (generic function with 1 method)

```
rot_Df2(X, Y) = inner(R,
  exp(R, rot_p, t*X),
  -log(R, exp(R, rot_p, t*X), rot_q),
  jacobi_field(R, rot_p, exp(R, rot_p, X), t, t * Y, βdifferential_exp_arg_with_time)
)
```




```
check_diff(TangentSpace(R, rot_p), rot_f, rot_Df2,
           p = random_tangent(R, rot_p),  v = random_tangent(R, rot_p))
```