In [ ]:

```
import torch
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

from scipy import stats
from torch.utils.data import DataLoader
from torchvision.datasets import FashionMNIST
```

Problem 1

a)

Before computing the formulas for the Christoffel symbols, we need to specify the following relationship:

$x^1 = \mu$

$x^2 = \sigma$

The summation goes from 1 to 2 because we have two variables (μ and σ), therefore we have:

$$\begin{aligned}\Gamma_{ij}^{k} &= \frac{1}{2}\sum_{l=1}^{n} g^{kl}\left(\frac{\partial g_{jl}}{\partial x^i} + \frac{\partial g_{il}}{\partial x^j} - \frac{\partial g_{ij}}{\partial x^l}\right) \\ &= \frac{1}{2}\left[g^{k1}\left(\frac{\partial g_{j1}}{\partial x^i} + \frac{\partial g_{i1}}{\partial x^j} - \frac{\partial g_{ij}}{\partial x^1}\right) + g^{k2}\left(\frac{\partial g_{j2}}{\partial x^i} + \frac{\partial g_{i2}}{\partial x^j} - \frac{\partial g_{ij}}{\partial x^2}\right)\right]\end{aligned}$$

Now we need to get g^{-1}, but we first need to get the determinant of g:

$$\begin{aligned}det(g) &= \frac{1}{ad - bc} \\ &= \frac{1}{\frac{1}{\sigma^2}.\frac{2}{\sigma^2} - 0.0} \\ &= \frac{1}{\frac{2}{\sigma^4}} \\ &= \frac{\sigma^4}{2}\end{aligned}$$

Now we can multiply the $det(g)$ by matrix g to obtaing g^{-1}:

$$\begin{aligned}g^{-1} &= det(g).\begin{bmatrix}\frac{2}{\sigma^2} & 0 \\ 0 & \frac{1}{\sigma^2}\end{bmatrix} \\ &= \frac{\sigma^4}{2}.\begin{bmatrix}\frac{2}{\sigma^2} & 0 \\ 0 & \frac{1}{\sigma^2}\end{bmatrix} \\ &= \begin{bmatrix}\sigma^2 & 0 \\ 0 & \frac{\sigma^2}{2}\end{bmatrix}\end{aligned}$$

From all partial derivatives, we know that only 2 are not going to be 0, and those are:

$\frac{\partial g_{11}}{\partial x^2} = \frac{-2}{\sigma^3}$

$\frac{\partial g_{22}}{\partial x^2} = \frac{-4}{\sigma^3}$

Now we are going to iterate over i, j and k to get all the Christoffel symbols and replace values using matrix g, g^{-1} and the partial derivatives calculated above:

$$\begin{aligned}\Gamma_{11}^{1} &= \frac{1}{2}\sum_{l=1}^{n} g^{1l}\left(\frac{\partial g_{1l}}{\partial x^1} + \frac{\partial g_{1l}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^l}\right) \\ &= \frac{1}{2}\left[g^{11}\left(\frac{\partial g_{11}}{\partial x^1} + \frac{\partial g_{11}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^1}\right) + g^{12}\left(\frac{\partial g_{12}}{\partial x^1} + \frac{\partial g_{12}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^2}\right)\right] \\ &= \frac{1}{2}\left[\sigma^2\left(0 + 0 - 0\right) + 0\right] \\ &= 0\end{aligned}$$

$$\begin{aligned}\Gamma_{11}^{2} &= \frac{1}{2}\sum_{l=1}^{n} g^{2l}\left(\frac{\partial g_{1l}}{\partial x^1} + \frac{\partial g_{1l}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^l}\right) \\ &= \frac{1}{2}\left[g^{21}\left(\frac{\partial g_{11}}{\partial x^1} + \frac{\partial g_{11}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^1}\right) + g^{22}\left(\frac{\partial g_{12}}{\partial x^1} + \frac{\partial g_{12}}{\partial x^1} - \frac{\partial g_{11}}{\partial x^2}\right)\right] \\ &= \frac{1}{2}\left[0 + \frac{\sigma^2}{2}\left(0 + 0 - \frac{-2}{\sigma^3})\right)\right] \\ &= \frac{1}{2}\left[\frac{2\sigma^2}{2\sigma^3}\right] \\ &= \frac{1}{2\sigma}\end{aligned}$$

$$
\begin{aligned}
\Gamma^1_{12} &= \frac{1}{2}\sum_{l=1}^{n} g^{1l}\left(\frac{\partial g_{2l}}{\partial x^1}+\frac{\partial g_{1l}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{11}\left(\frac{\partial g_{21}}{\partial x^1}+\frac{\partial g_{11}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^1}\right)+g^{12}\left(\frac{\partial g_{22}}{\partial x^1}+\frac{\partial g_{12}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[\sigma^2\left(0+\frac{-2}{\sigma^3}-0\right)+0\right]\\
&= \frac{1}{2}\left[\frac{-2\sigma^2}{\sigma^3}\right]\\
&= \frac{-1}{\sigma}
\end{aligned}
$$

$$
\begin{aligned}
\Gamma^2_{12} &= \frac{1}{2}\sum_{l=1}^{n} g^{2l}\left(\frac{\partial g_{2l}}{\partial x^1}+\frac{\partial g_{1l}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{21}\left(\frac{\partial g_{21}}{\partial x^1}+\frac{\partial g_{11}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^1}\right)+g^{22}\left(\frac{\partial g_{22}}{\partial x^1}+\frac{\partial g_{12}}{\partial x^2}-\frac{\partial g_{12}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[0+\frac{\sigma^2}{2}\left(0+0-0\right)\right]\\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\Gamma^1_{21} &= \frac{1}{2}\sum_{l=1}^{n} g^{1l}\left(\frac{\partial g_{1l}}{\partial x^2}+\frac{\partial g_{2l}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{11}\left(\frac{\partial g_{11}}{\partial x^2}+\frac{\partial g_{21}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^1}\right)+g^{12}\left(\frac{\partial g_{12}}{\partial x^2}+\frac{\partial g_{22}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[\sigma^2\left(\frac{-2}{\sigma^3}+0-0\right)+0\right]\\
&= \frac{1}{2}\left[\frac{-2\sigma^2}{\sigma^3}\right]\\
&= \frac{-1}{\sigma}
\end{aligned}
$$

$$
\begin{aligned}
\Gamma^2_{21} &= \frac{1}{2}\sum_{l=1}^{n} g^{2l}\left(\frac{\partial g_{1l}}{\partial x^2}+\frac{\partial g_{2l}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{21}\left(\frac{\partial g_{11}}{\partial x^2}+\frac{\partial g_{21}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^1}\right)+g^{22}\left(\frac{\partial g_{12}}{\partial x^2}+\frac{\partial g_{22}}{\partial x^1}-\frac{\partial g_{21}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[0+\frac{\sigma^2}{2}\left(0+0-0\right)\right]\\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\Gamma^1_{22} &= \frac{1}{2}\sum_{l=1}^{n} g^{1l}\left(\frac{\partial g_{2l}}{\partial x^2}+\frac{\partial g_{2l}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{11}\left(\frac{\partial g_{21}}{\partial x^2}+\frac{\partial g_{21}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^1}\right)+g^{12}\left(\frac{\partial g_{22}}{\partial x^2}+\frac{\partial g_{22}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[\sigma^2\left(0+0-0\right)+0\right]\\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
\Gamma^2_{22} &= \frac{1}{2}\sum_{l=1}^{n} g^{2l}\left(\frac{\partial g_{2l}}{\partial x^2}+\frac{\partial g_{2l}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^l}\right)\\
&= \frac{1}{2}\left[g^{21}\left(\frac{\partial g_{21}}{\partial x^2}+\frac{\partial g_{21}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^1}\right)+g^{22}\left(\frac{\partial g_{22}}{\partial x^2}+\frac{\partial g_{22}}{\partial x^2}-\frac{\partial g_{22}}{\partial x^2}\right)\right]\\
&= \frac{1}{2}\left[0+\frac{\sigma^2}{2}\left(\frac{-4}{\sigma^3}+\frac{-4}{\sigma^3}-\frac{-4}{\sigma^3}\right)\right]\\
&= \frac{1}{2}\left[\frac{\sigma^2}{2}\left(\frac{-4}{\sigma^3}\right)\right]\\
&= \frac{1}{2}\left[\frac{-2}{\sigma}\right]\\
&= \frac{-1}{\sigma}
\end{aligned}
$$

b)

To confirm that the curves satisfy the geodesic equation, we need to calculate $\frac{d\gamma^1}{dt}$ and $\frac{d\gamma^2}{dt}$

$\frac{d\gamma^1}{dt} = 0$
$\frac{d\gamma^2}{dt} = b.\,c.\,e^{c.t}$
$\frac{d^2\gamma^1}{dt^2} = 0$
$\frac{d^2\gamma^2}{dt^2} = b.\,c^2.\,e^{c.t}$

Now expand the summation and replace

$$\frac{d^2\gamma^1}{dt^2} = -\sum_{i,j=1}^{n} \Gamma^1_{ij}(\gamma(t))\frac{d\gamma^i}{dt}\frac{d\gamma^j}{dt}$$

$$0 = -\left[0 - \frac{1}{\sigma}\frac{d\gamma^1}{dt}\frac{d\gamma^2}{dt} - \frac{1}{\sigma}\frac{d\gamma^2}{dt}\frac{d\gamma^1}{dt} + 0\right]$$

$$0 = -\left[-\frac{1}{\sigma}.0.b.c.e^{c.t} - \frac{1}{\sigma}.b.c.e^{c.t}.0\right]$$

$$0 = -\left[0 - 0\right]$$

$$0 = 0$$

$$\frac{d^2\gamma^2}{dt^2} = -\sum_{i,j=1}^{n} \Gamma^2_{ij}(\gamma(t))\frac{d\gamma^i}{dt}\frac{d\gamma^j}{dt}$$

$$b.c^2.e^{c.t} = -\left[\frac{1}{2\sigma}\frac{d\gamma^1}{dt}\frac{d\gamma^1}{dt} + 0 + 0 - \frac{1}{\sigma}\frac{d\gamma^2}{dt}\frac{d\gamma^2}{dt}\right]$$

$$b.c^2.e^{c.t} = -\left[\frac{1}{2\sigma}.0.0 - \frac{1}{b.e^{c.t}}.b.c.e^{c.t}.b.c.e^{c.t}\right]$$

$$b.c^2.e^{c.t} = -\left[-b.c^2.e^{c.t}\right]$$

$$b.c^2.e^{c.t} = b.c^2.e^{c.t}$$

c)

To confirm that the curves satisfy the geodesic equation, we need to calculate $\frac{d\gamma^1}{dt}$ and $\frac{d\gamma^2}{dt}$

$\frac{d\gamma^1}{dt} = a.c.sech^2(c.t)$

$\frac{d\gamma^2}{dt} = \frac{-a.c.tanh(c.t).sech(c.t)}{\sqrt{2}}$

$\frac{d^2\gamma^1}{dt^2} = -2.a.c^2.tanh(c.t)sech^2(c.t)$

$\frac{d^2\gamma^2}{dt^2} = \frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}}$

Now expand the summation and replace

$$\frac{d^2\gamma^2}{dt^2} = -\sum_{i,j=1}^{n} \Gamma^k_{ij}(\gamma(t))\frac{d\gamma^i}{dt}\frac{d\gamma^j}{dt}$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -\left[0 - \frac{1}{\sigma}\frac{d\gamma^1}{dt}\frac{d\gamma^2}{dt} - \frac{1}{\sigma}\frac{d\gamma^2}{dt}\frac{d\gamma^1}{dt} + 0\right]$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -\left[0 - \frac{1}{\sigma}a.c.sech^2(c.t)\frac{-a.c.tanh(c.t).sech(c.t)}{\sqrt{2}} - \frac{1}{\sigma}\frac{-a.c.tanh(c.t).sech(c.t)}{\sqrt{2}}a.c.sech^2(c.t) + 0\right]$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -\left[0 - \frac{2}{a.\sqrt{2}.sech(c.t)}a.c.sech^2(c.t)\frac{(-a).c.tanh(c.t).sech(c.t)}{\sqrt{2}} - \frac{2}{a.\sqrt{2}.sech(c.t)}\frac{(-a).c.tanh(c.t).se}{\sqrt{2}}\right.$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -\left[sech^2(c.t).a.c^2.tanh(c.t) + sech^2(c.t).a.c^2.tanh(c.t)\right]$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -\left[2.sech^2(c.t).a.c^2.tanh(c.t)\right]$$

$$-2.a.c^2.tanh(c.t)sech^2(c.t) = -2.a.c^2.tanh(c.t)sech^2(c.t)$$

$$\frac{d^2\gamma^2}{dt^2} = -\sum_{i,j=1}^{n} \Gamma^2_{ij}(\gamma(t))\frac{d\gamma^i}{dt}\frac{d\gamma^j}{dt}$$

$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = -\left[\frac{1}{2\sigma}\frac{d\gamma^1}{dt}\frac{d\gamma^1}{dt} + 0 + 0 - \frac{1}{\sigma}\frac{d\gamma^2}{dt}\frac{d\gamma^2}{dt}\right]$$

$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = -\left[\frac{1}{2\sigma}a.c.sech^2(c.t)a.c.sech^2(c.t) - \frac{1}{\sigma}\frac{(-a).c.tanh(c.t).sech(c.t)}{\sqrt{2}}\frac{(-a).c.tanh(c.t).sech(c}{\sqrt{2}}\right.$$

$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = -\left[\frac{1}{2a.\frac{\sqrt{2}}{2}.sech(c.t)}a.c.sech^2(c.t)a.c.sech^2(c.t) - \frac{1}{a.\frac{\sqrt{2}}{2}.sech(c.t)}\frac{(-a).c.tanh(c.t).sech(c}{\sqrt{2}}\right.$$

$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = -\left[\frac{a.c^2.sech^3(c.t)}{\sqrt{2}} - \frac{a.c^2.tanh^2(c.t).sech(c.t)}{\sqrt{2}}\right]$$

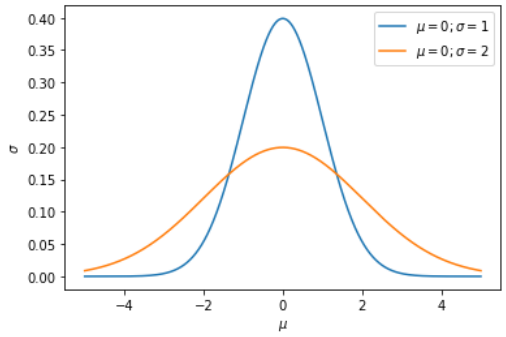
$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = -\frac{a.c^2.sech^3(c.t)}{\sqrt{2}} + \frac{a.c^2.tanh^2(c.t).sech(c.t)}{\sqrt{2}}$$

$$\frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}} = \frac{a.c^2.sech(c.t).\left[tanh^2(c.t) - sech^2(c.t)\right]}{\sqrt{2}}$$

The evaluation of the Christoffel symbols show that the left and right handside for both equation are the same, meaning that the curves of the form $(\mu(t), \sigma(t)) = (a.tanh(c.t) + b, a.\frac{\sqrt{2}}{2}.sech(c.t))$ satisfy the geodesic equation.

d)

In [ ]:
```
x_data = np.arange(-5, 5, 0.001)
y_data1 = stats.norm.pdf(x_data, 0, 1)
y_data2 = stats.norm.pdf(x_data, 0, 2)
plt.plot(x_data, y_data1, label="$\mu=0; \sigma=1$")
plt.plot(x_data, y_data2, label="$\mu=0; \sigma=2$")
plt.xlabel("$\mu$")
plt.ylabel("$\sigma$")
plt.legend();
```



In [ ]:
```
def geodesic_distance(x,y):
    y_aux = y[0]
    x_aux = x[0]
    distance = 0
    for i in range(1,x.shape[0]):
        # Metric matrix
        metric_matrix = np.array([[1/(y[i]**2), 0],[0, 2/(y[i]**2)]])

        # Deltas
        delta_x = x[i] - x_aux
        delta_y = y[i] - y_aux

        # Squared norm
        v = np.array([delta_x, delta_y])
        squared_norm = v.T @ metric_matrix @ v

        # Distance
        distance += np.sqrt(squared_norm)

        x_aux = x[i]
        y_aux = y[i]
    return distance
```

In [ ]:
```
def plot_geodesic_type1(a,b,c,t1,t2,step):
    """
    a,b,c: constants
    t1: start of segment
    t2: end of segment
    step: step size for increasing t
    """
    fig, ax = plt.subplots()
    # t = np.arange(t1,t2+step,step)
    t = np.arange(t1,t2,step)
    sigma = b * np.exp(c*t)
    mu = np.ones(t.shape[0]) * a
    distance = geodesic_distance(mu,sigma)

    print(f"Distance = {distance}")
    ax.plot(mu, sigma)
    ax.set_xlabel("$\mu$")
    ax.set_ylabel("$\sigma$")
```

Based on the equation provided in b), we know that $\mu = a$ and $\sigma = b.\, e^{c.t}$. If we use $t \in [0, 1]$ and solve for a, b and c, we obtain the following:

$a = 0$

$(\mu, \sigma) = (0, 1)$ for $t = 0$

$b.\, e^{c.0} = 1$
$b.\, e^{0} = 1$
$b.\, 1 = 1$
$b = 1$

$(\mu, \sigma) = (0, 2)$ for $t = 1$ and $b = 1$

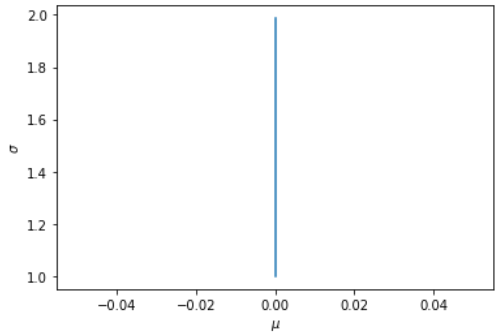
$b.\, e^{c.1} = 2$
$1.\, e^{c} = 2$
$e^{c} = 2$
$c = ln(2)$

What is the geodesic curve between them?

Geodesic curve between gaussians with (μ, σ) parameters: $(0, 1)$ and $(0, 2)$

In [ ]:
```
plot_geodesic_type1(0,1,np.log(2),0,1,0.01)
```

```
Distance = 0.9670999768746887
```



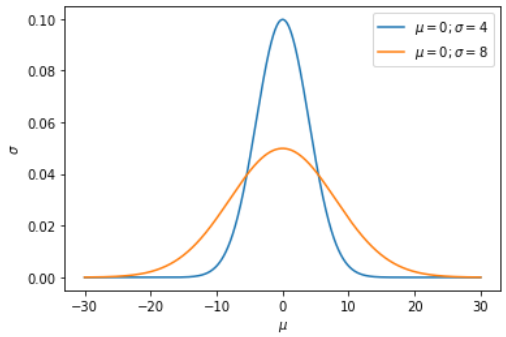
What is the geodesic distance between them?

The geodesic distance between the two gaussians is 0.967

d) Repeat questions for gaussians with (μ, σ) parameters: $(0, 4)$ and $(0, 8)$

pdf's

In [ ]:
```
x_data = np.arange(-30, 30, 0.001)
y_data1 = stats.norm.pdf(x_data, 0, 4)
y_data2 = stats.norm.pdf(x_data, 0, 8)
plt.plot(x_data, y_data1, label="$\mu=0; \sigma=4$")
plt.plot(x_data, y_data2, label="$\mu=0; \sigma=8$")
plt.xlabel("$\mu$")
plt.ylabel("$\sigma$")
plt.legend();
```

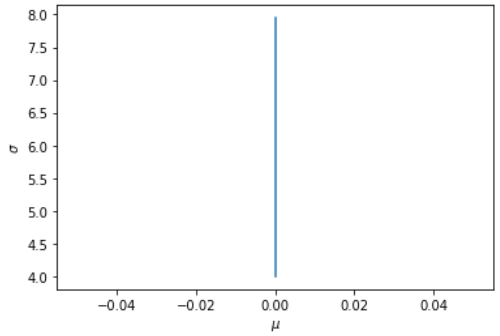


What is the geodesic curve between them?

Geodesic curve between gaussians with (μ, σ) parameters: $(0, 4)$ and $(0, 8)$

In [ ]:
```
plot_geodesic_type1(0,4,np.log(2),0,1,0.01)
```

```
Distance = 0.9670999768746887
```



What is the geodesic distance between them?

The geodesic distance between the two gaussians is 0.967

Can you conjecture a general rule about geodesic distances under scaling both standard deviations by the same factor?

Based on the results obtained, we can say that the geodesic distance between two gaussians distributions is **invariant** under scaling both standard

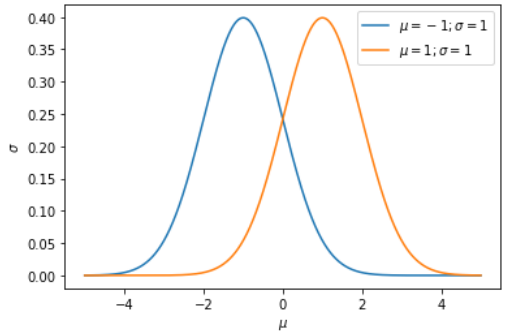
deviations by the same factor when using the Fisher Information metric. I.e. the geodesic distances are the same.

e)

Gaussians with (μ, σ) parameters: $(-1, 1)$ and $(1, 1)$ pdf's

In [ ]:

```
x_data = np.arange(-5, 5, 0.001)
y_data1 = stats.norm.pdf(x_data, -1, 1)
y_data2 = stats.norm.pdf(x_data, 1, 1)
plt.plot(x_data, y_data1, label="$\mu=-1; \sigma=1$")
plt.plot(x_data, y_data2, label="$\mu=1; \sigma=1$")
plt.xlabel("$\mu$")
plt.ylabel("$\sigma$")
plt.legend();
```



Based on the equation provided in c), we know that $\mu = a.tanh(c.t) + b$ and $\sigma = a.\frac{\sqrt{2}}{2}.sech(c.t)$. Then we solve for a:

$$\sigma = a.\frac{\sqrt{2}}{2}.sech(c.t)$$
$$\sigma = a.\frac{\sqrt{2}}{2}.\frac{1}{cosh(c.t)}$$
$$a = \frac{2.\sigma.cosh(c.t)}{\sqrt{2}}$$
$$a = \sqrt{2}.\sigma.cosh(c.t)$$

Solve for b:

$$\mu = a.tanh(c.t) + b$$
$$b = \mu - a.tanh(c.t)$$

Finally, solve for c. Note that given the trigonometry identity: $\tanh(x) = -\tanh(-x)$, we have:

$$\mu = a.tanh(c.t) + b$$
$$-\mu = a.tanh(-c.t) + b$$

then we can do the following:

$$\mu - (-\mu) = a.tanh(c.t) + b - (a.tanh(-c.t) + b)$$
$$\mu + \mu = a.tanh(c.t) + b - a.tanh(-c.t) - b$$
$$2.\mu = a.tanh(c.t) - a.tanh(-c.t)$$
$$2.\mu = a.tanh(c.t) + a.tanh(c.t)$$
$$2.\mu = 2.a.tanh(c.t)$$
$$\mu = a.tanh(c.t)$$
$$a = \frac{\mu}{tanh(c.t)}$$

if we equal our first equation for a and the above one:

$$\sqrt{2}.\sigma.cosh(c.t) = \frac{\mu}{tanh(c.t)}$$
$$cosh(c.t).tanh(c.t) = \frac{\mu}{\sqrt{2}.\sigma}$$
$$cosh(c.t).\frac{sinh(c.t)}{cosh(c.t)} = \frac{\mu}{\sqrt{2}.\sigma}$$
$$sinh(c.t) = \frac{\mu}{\sqrt{2}.\sigma}$$
$$c = arcsinh\left(\frac{\mu}{\sqrt{2}.\sigma}\right)$$

In [ ]:
```
def plot_geodesic_type2(a,b,c,t1,t2,step):
    """
    t1: start of segment
    t2: end of segment
    step: step size for increasing t
    """
    fig, ax = plt.subplots()

    t = np.arange(t1,t2,step)
    sigma = a * (np.sqrt(2) / 2) * (1 / np.cosh(c*t))
    mu = a * np.tanh(c*t) + b
    distance = geodesic_distance(mu,sigma)
    # middle_idx = int(t.shape[0] / 2 - 1)
    middle_idx = int(t.shape[0] / 2)

    # middle_t = 0
    # middle_sigma = a * (np.sqrt(2) / 2) * (1 / np.cosh(c*middle_t))
    # middle_mu = a * np.tanh(c*middle_t) + b

    print(f"Distance = {distance}")
    print(f"Middle point = ({mu[middle_idx]},{sigma[middle_idx]})")
    # print(f"Middle point = (\mu={middle_mu},\sigma={middle_sigma})")
    ax.plot(mu, sigma)
    ax.set_xlabel("$\mu$")
    ax.set_ylabel("$\sigma$")

    return (mu[middle_idx], sigma[middle_idx])
    # return (middle_mu, middle_sigma)
```

What is the geodesic curve between them?

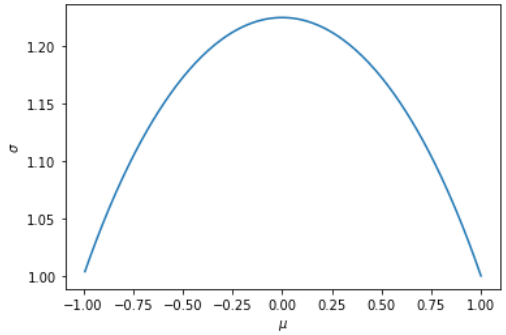
Geodesic curve between gaussians with (μ, σ) parameters: $(-1, 1)$ and $(1, 1)$

In [ ]:
```
_mu = -1
_sigma = 1

c = np.arcsinh(_mu / (np.sqrt(2) * _sigma) )
a = np.sqrt(2) * _sigma * np.cosh(c)
b = _mu - a * np.tanh(c)

middle_mu1, middle_sigma1 = plot_geodesic_type2(a,b,c,-1,1,0.01)
```

```
Distance = 1.8531343612629736
Middle point = (-1.3460512655173845e-15,1.2247448713915892)
```



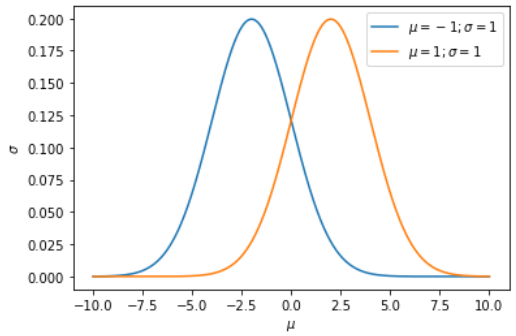
What is the geodesic distance between them?

The geodesic distance between the two gaussians is 1.853

e) Repeat questions for gaussians with (μ, σ) parameters: $(-2, 2)$ and $(2, 2)$

pdf's

In [ ]:
```
x_data = np.arange(-10, 10, 0.001)
y_data1 = stats.norm.pdf(x_data, -2, 2)
y_data2 = stats.norm.pdf(x_data, 2, 2)
plt.plot(x_data, y_data1, label="$\mu=-1; \sigma=1$")
plt.plot(x_data, y_data2, label="$\mu=1; \sigma=1$")
plt.xlabel("$\mu$")
plt.ylabel("$\sigma$")
plt.legend();
```



What is the geodesic curve between them?

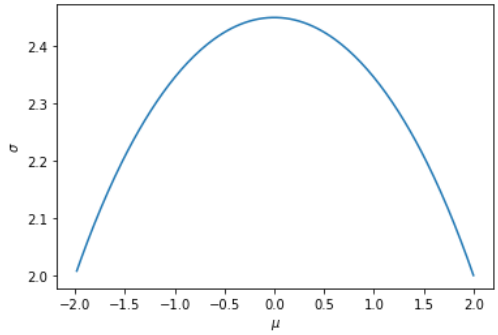
Geodesic curve between gaussians with (μ, σ) parameters: $(-2, 2)$ and $(2, 2)$

In [ ]:
```
_mu = -2
_sigma = 2

c = np.arcsinh(_mu / (np.sqrt(2) * _sigma) )
a = np.sqrt(2) * _sigma * np.cosh(c)
b = _mu - a * np.tanh(c)

middle_mu2, middle_sigma2 = plot_geodesic_type2(a,b,c,-1,1,0.01)
```

```
Distance = 1.8531343612629736
Middle point = (-2.692102531034769e-15,2.4494897427831783)
```



What is the geodesic distance between them?

The geodesic distance between the two gaussians is 1.853

Can you conjecture another general rule about geodesic distances?

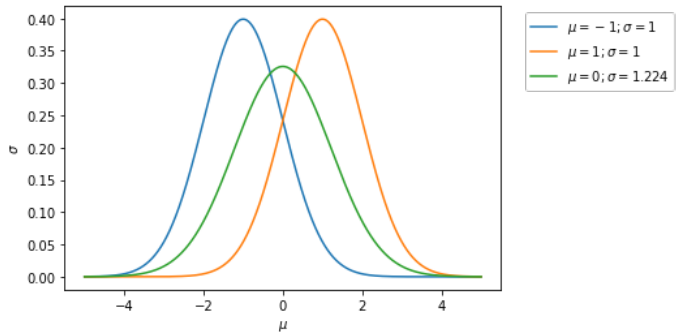
Based on the results obtained, we can say that the geodesic distance between two gaussians distributions is also **invariant** under scaling both means and standard deviations by the same factor when using the Fisher Information metric. I.e. the geodesic distances are the same.

f)

What is the Fréchet mean of these two? What is the midpoint (μ, σ) on the geodesic between them?

*The Fréchet mean is the green curve in the plot below

In [ ]:
```
x_data = np.arange(-5, 5, 0.001)
y_data1 = stats.norm.pdf(x_data, -1, 1)
y_data_middle = stats.norm.pdf(x_data, middle_mu1, middle_sigma1)
y_data2 = stats.norm.pdf(x_data, 1, 1)
plt.plot(x_data, y_data1, label="$\mu=-1; \sigma=1$")
plt.plot(x_data, y_data2, label="$\mu=1; \sigma=1$")
plt.plot(x_data, y_data_middle, label="$\mu=0; \sigma=1.224$")
plt.xlabel("$\mu$")
plt.ylabel("$\sigma$")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left");
```



What observations do you have about the mean?

If we take a look at the mean, we can see 2 things:

- The peak of the mean is in between the peak of the two gaussians
- The peak of the mean is lower than the peak of the two gaussians, which means that the mean distribution has a higher σ

Now if we take a look at the geodesic curve between the two gaussians with (μ, σ) parameters: $(-1, 1)$ and $(1, 1)$, we can see that the middle point in the curve is the one with highest σ. This means that for moving from the left gaussian distribution ($\mu = -1, \sigma = 1$) to the right gaussian distribution ($\mu = -1, \sigma = 1$), as the mean of the gaussian moves to the right, the standard deviation increase, until reaching the middle point, where the distribution has it's highest σ. From that point, if we keep moving the mean to the right, the standard deviation start to decrease until it reaches the final gaussian distribution.

Problem 2

In [ ]:
```
batch_size = 100
train_data = FashionMNIST("./data", train = True, download = True,
                          transform=transforms.ToTensor())
test_data = FashionMNIST("./data", train = False, download = True,
                          transform=transforms.ToTensor())

train_data_loader = DataLoader(train_data, batch_size = batch_size, shuffle = False)
test_data_loader = DataLoader(test_data, batch_size = batch_size, shuffle = False)
```

In [ ]:
```
def getClasses(classes, trainSet, testSet):
    train_data = []
    train_labels = []
    ### Train data
    for batch, labels in trainSet:
        for i in range(len(labels)):
            if labels[i] in classes:
                train_data.append(batch[i].view(784).tolist())
                train_labels.append(labels[i])
    ### Test data
    test_data = []
    test_labels = []
    for batch, labels in testSet:
        for i in range(len(labels)):
            if labels[i] in classes:
                test_data.append(batch[i].view(784).tolist())
                test_labels.append(labels[i])

    train_labels = torch.tensor(train_labels)
    test_labels = torch.tensor(test_labels)

    return torch.tensor(train_data), \
           torch.where(train_labels == 2, 1, 0), \
           torch.tensor(test_data), \
           torch.where(test_labels == 2, 1, 0)
```

In [ ]:
```
# MNIST Fashion class 2 - Pullover = 1
# MNIST Fashion class 7 - Sneaker = 0
train_data, train_labels, test_data, test_labels = getClasses([2,7], train_data_loader, test_data_loader)
```

In [ ]:

```
### Implementation based on https://www.geeksforgeeks.org/implementation-of-logistic-regression-from-scratch-using-python/

class LogisticRegression() :
    def __init__(self, learning_rate, iterations, weights, update_type="gd") :
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.W = weights
        self.update_type = update_type


    # Function for model training
    def fit(self, X, Y) :
        # no_of_training_examples, no_of_features
        self.m, self.n = X.shape
        # weight initialization
        # self.W = torch.rand(self.n)
        self.b = 0
        self.X = X
        self.Y = Y
        self.loss = []

        if self.update_type == 'gd':
        # gradient descent learning
            for i in range(self.iterations):
                self.update_weights_gd()
        elif self.update_type == 'ngd':
            for i in range(self.iterations):
                self.update_weights_ngd()
        return self


    # Update weights with gradient descent
    def update_weights_gd(self):
        # Make predictions
        prediction = 1 / (1 + torch.exp( - (self.X @ self.W + self.b) ) )

        # Epsilon to avoid numerical errors
        epsilon = 1e-4

        # Calculate gradients
        error = (prediction - self.Y.T).view(self.m)
        dW = self.X.T @ error / self.m  # 784x12000 @ 12000 = 784
        db = torch.sum(error) / self.m  # scalar

        # Calculate loss
        loss = -self.Y * torch.log(prediction+epsilon) - (1-self.Y) * torch.log(1-prediction+epsilon)
        self.loss.append(loss.sum().item() / self.m)

        # Update weights
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return self


    # Update weights with natural gradient
    def update_weights_ngd(self):
        # Make predictions
        prediction = 1 / (1 + torch.exp( - (self.X @ self.W + self.b) ) )

        # Epsilon to avoid numerical errors
        epsilon = 1e-4

        # Calculate gradients
        error = (prediction - self.Y.T).view(self.m)
        dW = self.X.T @ error / self.m  # 784x12000 @ 12000 = 784
        db = torch.sum(error) / self.m  # scalar

        # Calculate loss
        loss = -self.Y * torch.log(prediction+epsilon) - (1-self.Y) * torch.log(1-prediction+epsilon)
        self.loss.append(loss.sum().item() / self.m)

        ### Calculate natural gradient
        gradient = torch.zeros((784,784))

        # For each data point (image) in the dataset
        for i in range(self.m):
            aux = self.X[i] * error[i]  # 784 * scalar
            gradient += aux @ aux.T # 784x1 @ 1x784 = 784x784

        gradient = gradient / self.m    # 784x784 * scalar

        # Add epsilon avoid numerical error
        E = torch.eye(784) * epsilon    # 784x784
        gradient = gradient + E # 784x784 + 784x784 = 784x784

        # Calculate Fisher information
        F_INFORMATION = torch.inverse(gradient) # 784x784

        ### Update weights
        self.W = self.W - self.learning_rate * (F_INFORMATION @ dW)  # 784x784 @ 784x1 = 784x1
        self.b = self.b - self.learning_rate * db

        return self
```

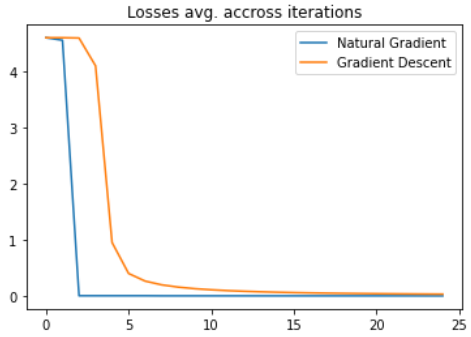
In [ ]:
```
losses = {}
for i in range(5):
    random_weights = torch.rand(train_data.shape[1])
    ngd_model = LogisticRegression(learning_rate=0.5, iterations=25, weights=random_weights, update_type='ngd')
    ngd_model.fit(train_data, train_labels)
    gd_model = LogisticRegression(learning_rate=0.5, iterations=25, weights=random_weights, update_type='gd')
    gd_model.fit(train_data, train_labels)
    iteration = {
        'ngd_loss':ngd_model.loss,
        'gd_loss':gd_model.loss
    }
    losses[str(i)] = iteration
```

In [ ]:
```
n_iterations = len(losses['0']['ngd_loss'])
x = [i for i in range(n_iterations)]

ngd_avg_loss = np.zeros(n_iterations)
gd_avg_loss = np.zeros(n_iterations)
for i in range(5):
    ngd_avg_loss += np.array(losses[str(i)]['ngd_loss'])
    gd_avg_loss += np.array(losses[str(i)]['gd_loss'])

ngd_avg_loss /= 5
gd_avg_loss /= 5

plt.plot(x,ngd_avg_loss, label="Natural Gradient")
plt.plot(x,gd_avg_loss, label="Gradient Descent")
plt.legend()
plt.title("Losses avg. accross iterations");
```



Which algorithm converges faster?

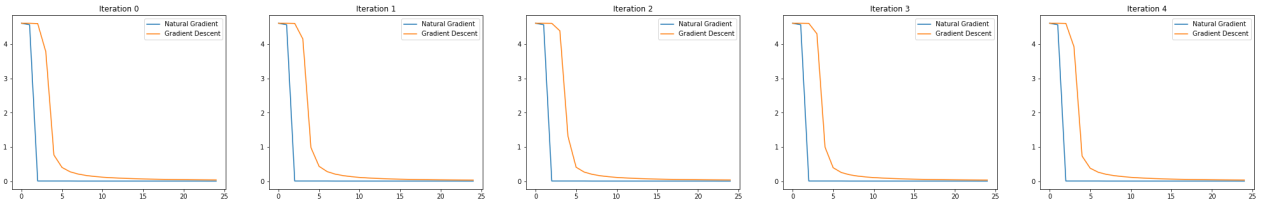
From the above plot, it can clearly be seen that the natural gradient algorithm converges 'faster', in terms of iterations, than the gradient descent algorithm. It's worth clarifying that the natural gradient converges in less iterations than the gradient descent, but not necessarily in less time, since the computation of the natural gradient is more expensive because it requires more operations. This may be one of the reasons that would explain why natural gradient is no widely used by the ML comunity as gradient descent.

In [ ]:
```
### Code for printing each iteration

n_iterations = len(losses['0']['ngd_loss'])
x = [i for i in range(n_iterations)]

# Plot config
fig, ax = plt.subplots(1,5)
fig.set_figheight(5)
fig.set_figwidth(35)

for i in range(5):
    ax[i].plot(x, losses[str(i)]['ngd_loss'], label="Natural Gradient")
    ax[i].plot(x, losses[str(i)]['gd_loss'], label="Gradient Descent")
    ax[i].set_title(f"Iteration {i}")
    ax[i].legend()
```



In [ ]: