In [1]:

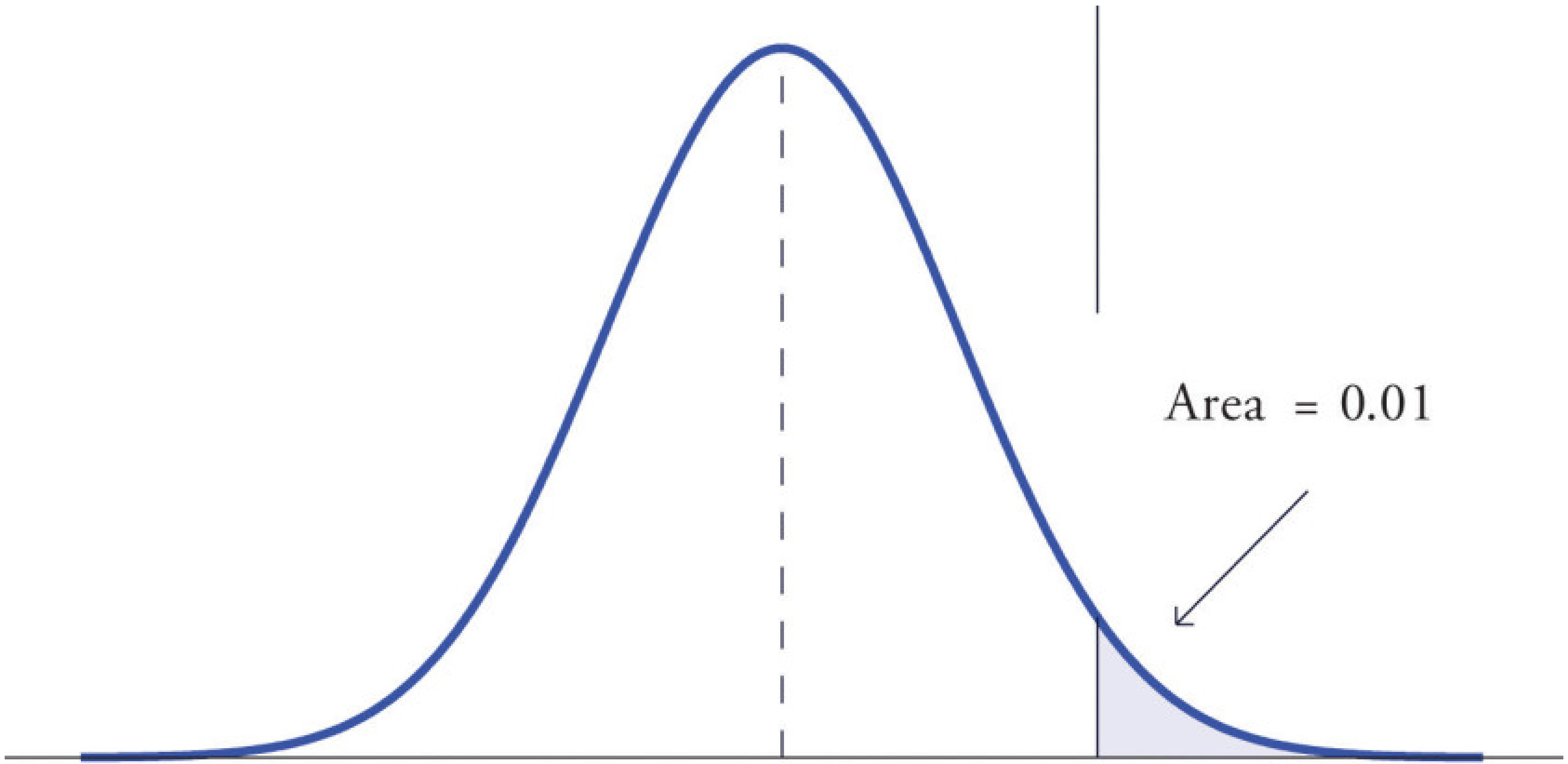
```python
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
#import qgrid
#qgrid.nbinstall(overwrite=True)

import numpy as np
import pandas as pd
```

The Conditional Value at Risk

https://en.wikipedia.org/wiki/Expected_shortfall

Thomas Schmelzer

The $\alpha = 0.99$ tail of a loss distribution



- In this talk we assume losses are postive. Larger losses, more pain... We want negative losses!
- The value at risk $\mathtt{VaR}_\alpha$ at level α is (the smallest) loss such that $\alpha\%$ of losses are smaller than $\mathtt{VaR}_\alpha$.
- This does not say anything about the magnitude of the losses larger than the $\mathtt{VaR}_\alpha$. We can only make statements about their number: $n(1-\alpha)$
- The $\mathtt{VaR}_\alpha$ has some sever mathematical flaws. It's not sub-additive, it's not convex. It's broken! However, the regulator embraced it.

- We compute the mean of the largest $n(1-\alpha)$ entries of a vector (or a optimal linear combination of vectors) without ever sorting the entries of any vector.
- The resulting convex program is linear.
- This mean is called Conditional Value at Risk $\mathtt{CVaR}_\alpha$ and is an upper bound for the Value at Risk $\mathtt{VaR}_\alpha$.

Given a vector $\mathbf{r}$ we introduce a free variable γ and define the function f as:

$$f(\gamma) = \gamma + \frac{1}{n(1-\alpha)} \sum (r_i - \gamma)^+$$

This is a continuous and convex function (in γ). The first derivative is:

$$f^{'}(\gamma) = 1 - \frac{\#\{r_i \geq \gamma\}}{n(1-\alpha)}$$

Given a vector $\mathbf{r}$ we introduce a free variable γ and define the function f as:

$$f(\gamma) = \gamma + \frac{1}{n\,(1-\alpha)} \sum (r_i - \gamma)^+$$

This is a continuous and convex function (in γ). The first derivative is:

$$f^{'}(\gamma) = 1 - \frac{\#\,\{r_i \geq \gamma\}}{n\,(1-\alpha)}$$

If γ such that $\#\{r_i \geq \gamma\} = n\,(1-\alpha)$:

- γ is a minimizer of f.
- $f(\gamma) = \mathtt{CVaR}_\alpha(\mathbf{r})$.

In [2]:

```
def f(gamma, returns, alpha=0.99):
    excess = returns - gamma
    return gamma + 1.0 / (len(returns) * (1 - alpha)) * excess[excess > 0].sum()

# note that cvar = (3+4)/2  and var = ? ... depends on your definition. 2?, 3?, 2.5?
r = np.array([-1.0, 2.0, 3.0, 2.0, 4.0, 2.0, 0.0, 1.0, -2.0, -2.0])
x = np.linspace(start=-1.0, stop=5.0, num=1000)
v = np.array([f(gamma=g, returns=r, alpha=0.80) for g in x])

plt.plot(x, v), plt.grid(True), plt.xlabel('$\gamma$'), plt.ylabel('$f$')
plt.title('Conditional value at risk as global minimum of a function f')
plt.axis([0, 5, 3, 6])
plt.show()
```

Before (using conic reformulation of the x^+ function):

- $$\mathtt{CVaR}(\mathbf{r}) = \min_{\gamma \in \mathbb{R}, \mathbf{t} \in \mathbb{R}^n} \gamma + \frac{1}{n(1-\alpha)} \sum t_i$$
  $$\text{s.t. } t_i \geq r_i - \gamma$$
  $$\mathbf{t} \geq 0$$

Now

- http://www.cvxpy.org/en/latest/tutorial/functions/, in particular the $x^+ = \max\{0, x\}$

In [3]:

```python
from cvx.util import minimize, cvx

R = [-1.0, 2.0, 3.0, 2.0, 4.0, 2.0, 0.0, 1.0, -2.0, -2.0]

n = len(R)
# We are interested in CVaR for alpha=0.80, e.g. what's the mean of the 20% of the b
alpha = 0.80

# introduce the variable for the var
gamma = cvx.Variable(1)
cvar = minimize(objective=gamma + 1.0/(n*(1-alpha)) * cvx.sum(cvx.pos(R - gamma)))

print("A minimizer of f (<= VaR):  {0}".format(gamma.value))
print("Minimum of f (== CVaR):     {0}".format(cvar))
```

```
A minimizer of f (<= VaR):  [2.33333333]
Minimum of f (== CVaR):     3.5000000000000004
```

In [4]:

```
from cvx.util import minimize
# take some random return data
R = np.random.randn(2500,100)
n,m = R.shape

# We are interested in CVaR for alpha=0.95, e.g. what's the mean of the 5% of the bi

alpha = 0.95

gamma, w = (cvx.Variable(1), cvx.Variable(m))
obj = gamma + 1.0/(n*(1-alpha)) * cvx.sum(cvx.pos(R*w - gamma))
cvar = minimize(objective=obj, constraints=[0 <= w, cvx.sum(w) == 1])
weights = w.value

plt.hist(R @ weights, bins=100)
plt.axis([-0.4, 0.4, 0, 150])
plt.title("CVaR {0}".format(cvar))
plt.show()
```

Summary

- We could compute the $\mathtt{CVaR}$ for a vector of length n by solving a linear program.
- We do not need to sort the elements nor do we need to know the Value at Risk $\mathtt{VaR}$.

In practice the vector $\mathbf{r}$ is not given. Rather we have m assets and try to find a linear combination of their corresponding return vectors such that the resulting portfolio has minimal Conditional Value at Risk.

```
In [ ]:
```