2.

$$E[U(x)] = E\left[x - \frac{ax^2}{2}\right] = E[x] - \frac{a}{2}E[x^2]$$

$$= \mu - \frac{a}{2}\left(\mathrm{Var}(x) + E[x]^2\right) = \boxed{\mu - \frac{a}{2}(\sigma^2 + \mu^2)}$$

X_{CE} s.t. $U(X_{CE}) = E[U(x)] = \mu - \frac{a}{2}(\sigma^2 + \mu^2)$

$$X_{CE} - \frac{aX_{CE}^2}{2} = \mu - \frac{a}{2}(\sigma^2 + \mu^2)$$

$$-\frac{a}{2}X_{CE}^2 + X_{CE} - \left(\mu - \frac{a}{2}(\sigma^2 + \mu^2)\right) = 0$$

$$X_{CE} = \frac{1 \pm \sqrt{1 - 2a\left(\mu - \frac{a}{2}(\sigma^2 + \mu^2)\right)}}{a}$$

$$\pi_A = E[x] - X_{CE} = \mu - \frac{1 \pm \sqrt{1 - 2a\left(\mu - \frac{a}{2}(\sigma^2 + \mu^2)\right)}}{a}$$

$$\approx -\frac{1}{2}\frac{U''(\mu)}{U'(\mu)}\sigma^2 = \frac{1}{2}\frac{a}{1-a\mu}\sigma^2 \qquad \frac{-U''(x)}{U'(x)} = \frac{a}{1-ax}$$

$A(x) = \frac{a}{1-ax}$ $\qquad X_{CE} \approx \bar{X} - \pi_A = \mu - \frac{1}{2}\frac{a}{1-a\mu}\sigma^2$

$\pi = \frac{2}{10^6}$ $\qquad W \sim N(1 + r + \pi(\mu - r), \pi^2\sigma^2)$

Maximize $E[U(W)] = M_W - \frac{a}{2}(\sigma_W^2 + \mu_W^2)$

$$= 1 + r + \pi(\mu - r) - \frac{a}{2}\left(\pi^2\sigma^2 + (1 + r + \pi(\mu - r))^2\right) \quad W = 1 + r + \pi(\mu - r)$$

$$\frac{\partial E[U(W)]}{\partial \pi} = \mu - r - \sigma^2 a\pi - a(1 + r + \pi(\mu - r))(\mu - r) = 0$$

$$\mu - r - \sigma^2 a\pi - a(1+r)(\mu - r) - a\pi(\mu - r)^2 = 0$$

$$\mu - r - a(1+r)(\mu - r) = \sigma^2 a\pi - a\pi(\mu - r)^2$$

$$\mu - r - a(1+r)(\mu - r) = \pi\left(a\left(\sigma^2 - (\mu - r)^2\right)\right)$$

$$\pi^* = \frac{\mu - r - a(1+r)(\mu - r)}{a\left(\sigma^2 - (\mu - r)^2\right)} \Rightarrow z^* = 10^6\pi^*$$

with $\mu = 0.2$, $r = 0.05$, $\sigma^2 = 0.01$

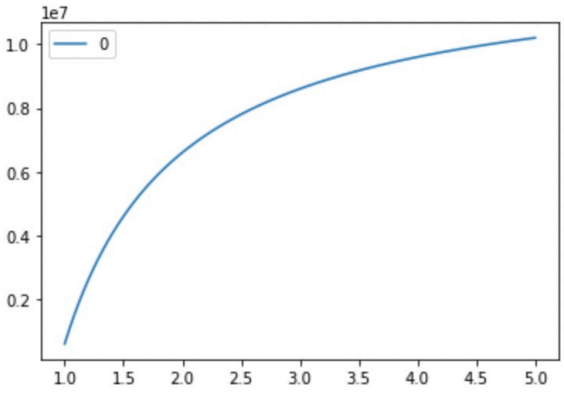
In [43]:
```
import numpy as np
import pandas as pd
```

In [51]:
```
mu = 0.2
r = 0.05
sigma2 = 0.01
z = [1e6 * (mu - r - a * (1 + r) * (mu - r)) / (a * (sigma2 - (mu - r) ** 2)) for alpha in np.linspace(1, 5, 100)]
```

In [47]:
```
plot_df = pd.DataFrame(z_list, index=alpha)
```

In [48]:
```
plot_df.plot()
```

Out[48]: <AxesSubplot:>



3.

- $W_1 = W_0(1 + f(1+\alpha))$
- $W_1 = W_0(1 + f(1-\beta))$
- $U(W_1) = \log(W_0) + \log(1 + f(1+\alpha))$
- $U(W_1) = \log(W_0) + \log(1 + f(1-\beta))$
- $E[\log W_1] = p[\log W_0 + \log(1 + f(1+\alpha))] + (1-p)(\log W_0 + \log(1 + f(1-\beta)))$

$= \log W_0 + p\log(1 + f(1+\alpha)) + (1-p)\log(1 + f(1-\beta))$

- $\frac{\partial E[\log W_1]}{\partial f} = \frac{p(1+\alpha)}{1 + f(1+\alpha)} + \frac{(1-p)(1-\beta)}{1 + f(1-\beta)}$
- $\frac{\partial E[\log(W_1)]}{\partial f} = 0 \Rightarrow \frac{p(1+\alpha)}{1 + f(1+\alpha)} = \frac{-(1-p)(1-\beta)}{1 + f(1-\beta)}$

$(1 + f(1-\beta))\, p(1+\alpha) = -(1 + f(1+\alpha))(1-p)(1-\beta)$

$p(1+\alpha) + (1-p)(1-\beta) = -f(1+\alpha)(1-p)(1-\beta) - f(1-\beta)p(1+\alpha)$

$p(1+\alpha) + (1-p)(1-\beta) = -f[(1+\alpha)(1-p)(1-\beta) + (1+\alpha)p(1-\beta)]$

$f^* = \frac{-p(1+\alpha) - (1-p)(1-\beta)}{(1+\alpha)(1-\beta)}$

$f^* = \frac{-p}{1-\beta} - \frac{(1-p)}{1+\alpha}$

$\frac{\partial^2 E\log W_1}{\partial f^2} = \frac{-p(1+\alpha)^2}{[1 + f(1+\alpha)]^2} - \frac{(1-p)(1-\beta)^2}{(1 + f(1-\beta))^2}$

evaluate for $f = f^*$: $\frac{-p(1+\alpha)^2}{\left(1 + \left[\frac{-p}{1-\beta} - \frac{(1-p)}{1+\alpha}\right](1+\alpha)\right)^2} - \frac{(1-p)(1-\beta)^2}{\left(1 + \left(\frac{-p}{1-\beta} - \frac{(1-p)}{1+\alpha}\right)(1-\beta)\right)^2}$

$\frac{(-)}{(+)} - \frac{(+)}{(+)} = (-)$, so f^* indeed a maxima

- $f^* \uparrow$ as $\alpha \uparrow$; increase wager when gain potential increases
- $f^* \uparrow$ as $\beta \downarrow$: increase wager when loss potential decreases
- $f^* \uparrow$ as $p \uparrow$ (if $\beta > 1$): increase wager as probability for gain increases

1.

In [112]:
```
from typing import Iterator, Tuple, TypeVar, Sequence, List, Dict
from operator import itemgetter
import numpy as np
import os
import sys
sys.path.append(os.path.abspath("/Users/justincramer/Documents/Coding/CME241/RL-book/"))
import itertools
from rl.distribution import Distribution, Choose
from rl.function_approx import FunctionApprox, Tabular
from rl.iterate import iterate
from rl.markov_process import (FiniteMarkovRewardProcess, MarkovRewardProcess,
                               RewardTransition, NonTerminal, State)
from rl.markov_decision_process import (FiniteMarkovDecisionProcess,
                                        MarkovDecisionProcess,
                                        StateActionMapping)
from rl.policy import DeterministicPolicy, FinitePolicy, FiniteDeterministicPolicy
import rl.approximate_dynamic_programming
from rl.approximate_dynamic_programming import evaluate_mrp, value_iteration, extended_vf, evaluate_finite_mrp
from rl.dynamic_programming import policy_iteration as exact_policy_iteration
```

In [37]:
```
S = TypeVar('S')
A = TypeVar('A')

# A representation of a value function for a finite MDP with states of
# type S
ValueFunctionApprox = FunctionApprox[NonTerminal[S]]
QValueFunctionApprox = FunctionApprox[Tuple[NonTerminal[S], A]]
NTStateDistribution = Distribution[NonTerminal[S]]
```

In [98]:
```
# approximate policy iteration
def policy_iteration(
    mdp: MarkovDecisionProcess[S, A],
    γ: float,
    approx_0: Tuple[ValueFunctionApprox[S], FinitePolicy[S, A]],
    non_terminal_states_distribution: NTStateDistribution[S],
    num_state_samples: int
) -> Iterator[Tuple[ValueFunctionApprox[S], FinitePolicy[S, A]]]:

    def update(vf_policy: Tuple[ValueFunctionApprox[S], FinitePolicy[S, A]])\
            -> Tuple[ValueFunctionApprox[S], FiniteDeterministicPolicy[S, A]]:
        vf, pi = vf_policy
        # apply policy to get mrp
        mrp: FiniteMarkovRewardProcess[S] = mdp.apply_finite_policy(pi)
        vf_iter: Iterator[ValueFunctionApprox[S]] = evaluate_mrp(mrp, γ, vf,
                                                                 non_terminal_states_distribution,
                                                                 num_state_samples)
        vf: ValueFunctionApprox[S] = list(itertools.islice(vf_iter, 200))[-1]

        nt_states: Sequence[NonTerminal[S]] = \
            non_terminal_states_distribution.sample_n(num_state_samples)

        def return_(s_r: Tuple[State[S], float]) -> float:
            s1, r = s_r
            return r + γ * extended_vf(vf, s1)


        d: Dict[S, A] = {s: max(mdp.actions(s),
                                key=lambda a: mdp.step(s, a).expectation(return_)) for s in nt_states}

        pi: FiniteDeterministicPolicy[S, A] = FiniteDeterministicPolicy(d)

        return vf, pi

    v_0, pi_0 = approx_0
    return iterate(update, (v_0, pi_0))
```