Exact Value Function for Recursive Utility

Lorenzo Naranjo

March 2026

Introduction

The Affine Recursive Utility in Continuous Time notebook derived the exact nonlinear ODE for the log continuation-value function

$$q(x) = \ln h(x),$$

in the one-factor Gaussian endowment economy

$$d \ln c_t = (\mu_0 + \mu_1 x_t)\, dt + \sigma_c\, dB_t, \qquad dx_t = \xi(\bar{x} - x_t)\, dt + \sigma_x\, dB_t.$$

That notebook then switched to a local affine approximation because it delivers closed-form intuition. This notebook goes the other way: it solves the exact boundary-value problem numerically and uses it to assess the affine approximation.

For one benchmark calibration, how much does the exact nonlinear solution differ from the affine approximation? Everything below is organized around that question. First we solve the boundary-value problem, then we recover the implied equilibrium objects, and finally we compare them with the affine benchmark.

With the homothetic value function

$$V(W, x) = \frac{W^{1-\gamma}}{1-\gamma} h(x),$$

the exact nonlinear ODE is

$$0 = \theta\left(\delta^{\psi} e^{-(\psi/\theta)q(x)} - \delta\right) + (1-\gamma)(\mu_0 + \mu_1 x) + \psi\xi(\bar{x} - x)q'(x)$$

$$+\frac{1}{2}\psi\sigma_x^2 q''(x) + \psi(1-\gamma)\sigma_c\sigma_x q'(x) + \frac{1}{2}\psi^2\sigma_x^2 q'(x)^2 + \frac{1}{2}(1-\gamma)^2\sigma_c^2, \qquad (1)$$

where

$$\theta = \frac{1-\gamma}{1-1/\psi}.$$

The boundary conditions are

$$q'(x_{\min}) = 0, \qquad q'(x_{\max}) = 0,$$

which approximate the tail conditions $q'(x) \to 0$ as $x \to \pm\infty$ on a finite interval.

Once $q(x)$ is known, the main equilibrium objects follow immediately:

$$m(x) = \delta^{\psi} e^{-(\psi/\theta)q(x)},$$

$$\sigma_W(x) = \sigma_c + \frac{\psi}{\theta} q'(x)\sigma_x,$$

and

$$\mu_W(x) = \mu_0 + \mu_1 x + \frac{\psi}{\theta} q'(x)\xi(\bar{x} - x) + \frac{1}{2}\frac{\psi}{\theta} q''(x)\sigma_x^2 + \frac{1}{2}\sigma_W(x)^2.$$

Calibration

The numerical solution needs a concrete parameterization. The values below are chosen to make the recursive-utility channel visible while keeping the boundary-value problem well behaved on a compact grid. The goal is not a broad sensitivity analysis, but a single clean benchmark against which to judge the affine approximation.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from scipy.integrate import solve_bvp
from scipy.optimize import root_scalar
```

```
params = {
    "delta": 0.02,
    "gamma": 8.0,
    "psi": 1.5,
    "mu0": 0.015,
    "mu1": 0.02,
    "xi": 0.35,
    "xbar": 0.0,
    "sigma_c": 0.02,
    "sigma_x": 0.06,
}

delta = params["delta"]
gamma = params["gamma"]
psi = params["psi"]
theta = (1 - gamma) / (1 - 1 / psi)
alpha = psi / theta

param_table = pd.Series(
    {
        " ": delta,
        " ": gamma,
        " ": psi,
        " ": theta,
        "  ": params["mu0"],
        "  ": params["mu1"],
        " ": params["xi"],
        "x̄": params["xbar"],
        " _c": params["sigma_c"],
        " _x": params["sigma_x"],
    }
)
```

```
param_table.to_frame("Value")
```

	Value
δ	0.020
γ	8.000
ψ	1.500
θ	-21.000
μ_0	0.015
μ_1	0.020
ξ	0.350
$\bar{x}$	0.000
σ_c	0.020
σ_x	0.060

Solving the Boundary-Value Problem

Write $y_1(x) = q(x)$ and $y_2(x) = q'(x)$. Then (1) becomes a first-order system:

$$y_1'(x) = y_2(x),$$

$$y_2'(x) = -\frac{2}{\psi\sigma_x^2}\Big[\theta\big(\delta^{\psi}e^{-\alpha y_1(x)} - \delta\big) + (1-\gamma)(\mu_0 + \mu_1 x) + \psi\xi(\bar{x} - x)y_2(x)$$

$$+\psi(1-\gamma)\sigma_c\sigma_x y_2(x) + \frac{1}{2}\psi^2\sigma_x^2 y_2(x)^2 + \frac{1}{2}(1-\gamma)^2\sigma_c^2\Big].$$

We solve this system on the truncated interval $[x_{\min}, x_{\max}] = [-0.12, 0.12]$ using `scipy.integrate.solve_bvp`. As an initial guess, we use the deterministic steady state obtained by setting $x = \bar{x}$ and imposing $q'(\bar{x}) = q''(\bar{x}) = 0$.

```
x_min, x_max = -0.12, 0.12
x_grid = np.linspace(x_min, x_max, 401)

steady_term = (
    (1 - gamma) * (params["mu0"] + params["mu1"] * params["xbar"])
    + 0.5 * (1 - gamma) ** 2 * params["sigma_c"] ** 2
)
m_steady = delta - steady_term / theta
q_steady = -(1 / alpha) * np.log(m_steady / delta**psi)


def recursive_utility_ode(x, y):
    q = y[0]
    q_prime = y[1]

    forcing = (
        theta * (delta**psi * np.exp(-alpha * q) - delta)
        + (1 - gamma) * (params["mu0"] + params["mu1"] * x)
        + psi * params["xi"] * (params["xbar"] - x) * q_prime
        + psi * (1 - gamma) * params["sigma_c"] * params["sigma_x"] * q_prime
        + 0.5 * psi**2 * params["sigma_x"] ** 2 * q_prime**2
        + 0.5 * (1 - gamma) ** 2 * params["sigma_c"] ** 2
    )

    q_second = -2 * forcing / (psi * params["sigma_x"] ** 2)
    return np.vstack((q_prime, q_second))


def boundary_conditions(left, right):
    return np.array([left[1], right[1]])
```

```
initial_q = q_steady + 0.15 * (x_grid - params["xbar"])
initial_q_prime = np.full_like(x_grid, 0.15)
initial_guess = np.vstack((initial_q, initial_q_prime))

solution = solve_bvp(
    recursive_utility_ode,
    boundary_conditions,
    x_grid,
    initial_guess,
    tol=1e-6,
    max_nodes=20_000,
)

print(solution.message)
print(f"Solver status: {solution.status}")
print(f"Mesh points used: {solution.x.size}")
```

```
The algorithm converged to the desired accuracy.
Solver status: 0
Mesh points used: 401
```

To analyze the solution, evaluate it on a fine grid and recover $q''(x)$ from the ODE rather than from numerical differentiation.

```
x_plot = np.linspace(x_min, x_max, 801)
q_exact, q_prime_exact = solution.sol(x_plot)

forcing_exact = (
    theta * (delta**psi * np.exp(-alpha * q_exact) - delta)
    + (1 - gamma) * (params["mu0"] + params["mu1"] * x_plot)
    + psi * params["xi"] * (params["xbar"] - x_plot) * q_prime_exact
    + psi * (1 - gamma) * params["sigma_c"] * params["sigma_x"] * q_prime_exact
```

```
    + 0.5 * psi**2 * params["sigma_x"] ** 2 * q_prime_exact**2
    + 0.5 * (1 - gamma) ** 2 * params["sigma_c"] ** 2
)
q_second_exact = -2 * forcing_exact / (psi * params["sigma_x"] ** 2)

m_exact = delta**psi * np.exp(-alpha * q_exact)
sigma_w_exact = params["sigma_c"] + alpha * q_prime_exact * params["sigma_x"]
mu_w_exact = (
    params["mu0"]
    + params["mu1"] * x_plot
    + alpha * q_prime_exact * params["xi"] * (params["xbar"] - x_plot)
    + 0.5 * alpha * q_second_exact * params["sigma_x"] ** 2
    + 0.5 * sigma_w_exact**2
)
```

The next table summarizes the range of the exact solution over the state grid.

```
summary = pd.DataFrame(
    {
        "Minimum": [
            q_exact.min(),
            q_prime_exact.min(),
            q_second_exact.min(),
            m_exact.min(),
            sigma_w_exact.min(),
            mu_w_exact.min(),
        ],
        "Maximum": [
            q_exact.max(),
            q_prime_exact.max(),
            q_second_exact.max(),
            m_exact.max(),
            sigma_w_exact.max(),
```

```
            mu_w_exact.max(),
        ],
    },
    index=[
        "q(x)",
        "q (x)",
        "q (x)",
        "m(x)",
        " _W(x)",
        " _W(x)",
    ],
)

summary.round(6)
```

	Minimum	Maximum
q(x)	23.860670	23.895639
q'(x)	-0.193475	0.000000
q"(x)	-5.272369	6.869608
m(x)	0.015550	0.015589
σ_W(x)	0.020000	0.020829
μ_W(x)	0.013478	0.016717

Comparing Exact and Affine Solutions

The affine approximation from the earlier notebook provides the benchmark. The point here is not to re-derive that approximation, but to measure what it misses relative to the exact nonlinear solution.

Under that approximation,

$$q_{\text{aff}}(x) = a + bx.$$

Its coefficients solve

$$\psi b(\xi + \bar{m}) = (1 - \gamma)\mu_1$$

and

$$\delta\theta = \theta\bar{m} + (1 - \gamma)(\mu_0 + \mu_1\bar{x}) + \frac{1}{2}(1 - \gamma)^2\sigma_w^2 + \frac{1}{2}b^2\sigma_x^2 + (1 - \gamma)b\sigma_w\sigma_x,$$

where

$$\sigma_w = \sigma_c + \frac{\psi}{\theta}b\sigma_x.$$

This leaves a single nonlinear equation in $\bar{m}$, which we solve numerically and then use to recover a and b.

```
def affine_residual(m_bar):
    b = (1 - gamma) * params["mu1"] / (psi * (params["xi"] + m_bar))
    sigma_w = params["sigma_c"] + alpha * b * params["sigma_x"]

    return (
        theta * m_bar
        + (1 - gamma) * (params["mu0"] + params["mu1"] * params["xbar"])
        + 0.5 * (1 - gamma) ** 2 * sigma_w**2
        + 0.5 * b**2 * params["sigma_x"] ** 2
        + (1 - gamma) * b * sigma_w * params["sigma_x"]
        - delta * theta
    )


m_bar = root_scalar(affine_residual, bracket=[1e-4, 0.05]).root
b = (1 - gamma) * params["mu1"] / (psi * (params["xi"] + m_bar))
a = theta * np.log(delta) - (theta / psi) * np.log(m_bar) - b * params["xbar"]

q_affine = a + b * x_plot
m_affine = delta**psi * np.exp(-alpha * q_affine)
sigma_w_affine = params["sigma_c"] + alpha * b * params["sigma_x"]
```

```
q_error = q_exact - q_affine
m_error = m_exact - m_affine
sigma_w_error = sigma_w_exact - sigma_w_affine

comparison = pd.Series(
    {
        "m̄": m_bar,
        "a": a,
        "b": b,
        " _w": sigma_w_affine,
        "max_x |q(x) - q_aff(x)|": np.max(np.abs(q_exact - q_affine)),
        "max_x |m(x) - m_aff(x)|": np.max(np.abs(m_exact - m_affine)),
    }
)

comparison.to_frame("Value")
```

	Value
m̄	0.015632
a	23.934742
b	-0.255265
σ_w	0.021094
max_x |q(x) - q_aff(x)|	0.069735
max_x |m(x) - m_aff(x)|	0.000078

The affine approximation is not exact, but the discrepancy is easy to quantify directly rather than judge only from the plots.

```
error_summary = pd.DataFrame(
    {
        "Max abs. error": [
```

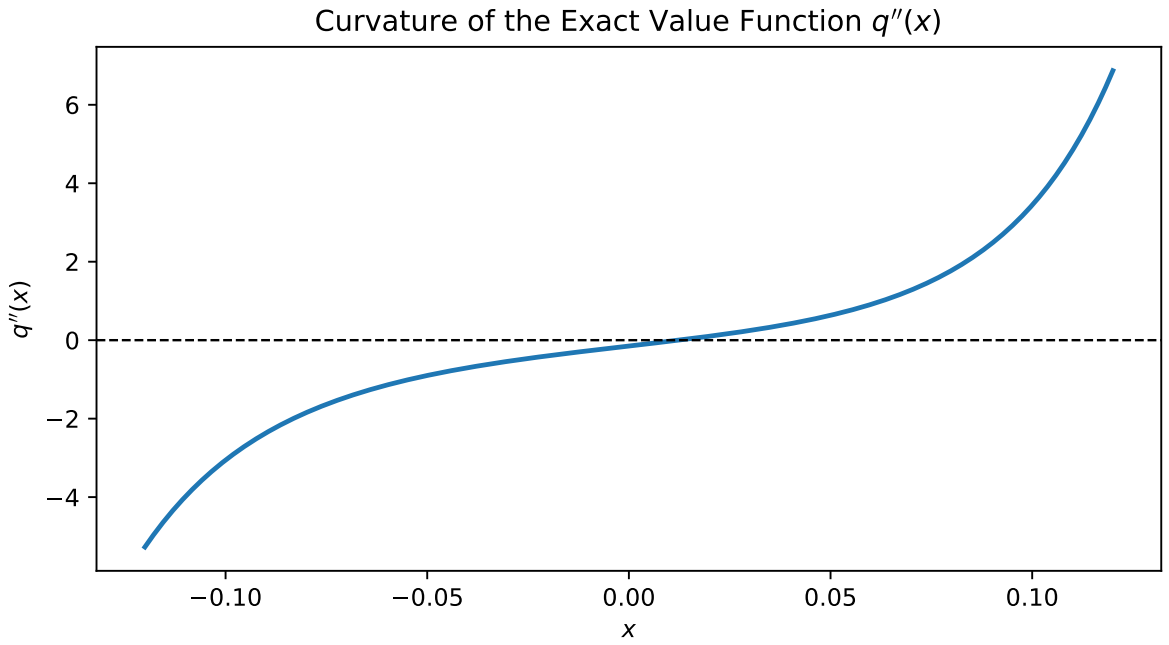
```
            np.max(np.abs(q_error)),
            np.max(np.abs(m_error)),
            np.max(np.abs(sigma_w_error)),
        ],
        "RMS error": [
            np.sqrt(np.mean(q_error**2)),
            np.sqrt(np.mean(m_error**2)),
            np.sqrt(np.mean(sigma_w_error**2)),
        ],
    },
    index=[
        "q(x)",
        "m(x)",
        " _W(x)",
    ],
)

error_summary.round(8)
```

	Max abs. error	RMS error
q(x)	0.069735	0.056475
m(x)	0.000078	0.000063
σ_W(x)	0.001094	0.000520

The cleanest way to see what the affine approximation misses is to look directly at the curvature of the exact solution. Because the affine specification imposes $q''(x) = 0$, any nonzero curvature is a genuinely nonlinear feature of the Epstein-Zin problem.

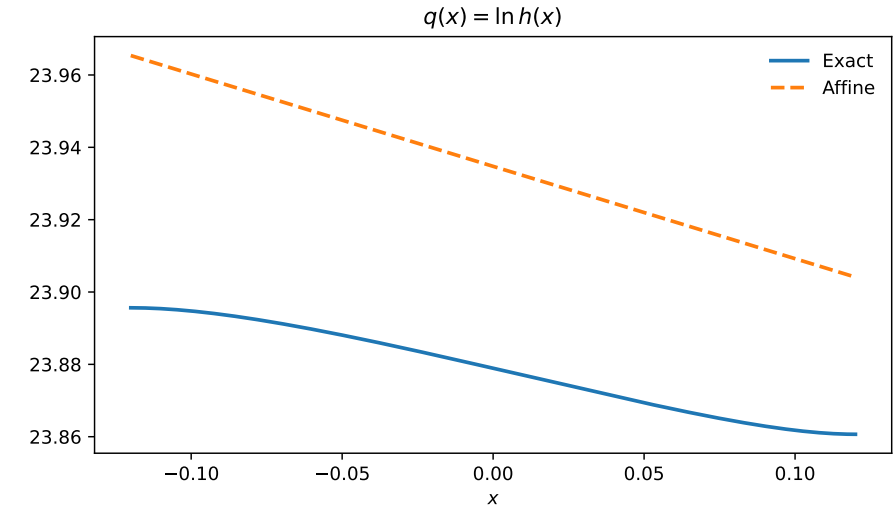
```
plt.figure(figsize=(7, 4))
plt.plot(x_plot, q_second_exact, lw=2)
plt.axhline(0.0, color="black", ls="--", lw=1)
```

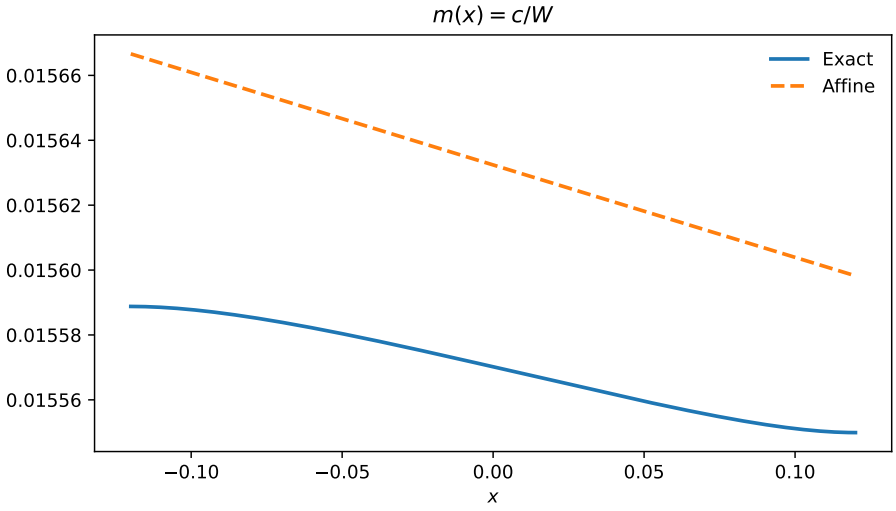
```
plt.title(r"Curvature of the Exact Value Function $q''(x)$")
plt.xlabel(r"$x$")
plt.ylabel(r"$q''(x)$")
plt.tight_layout()
plt.show()
```

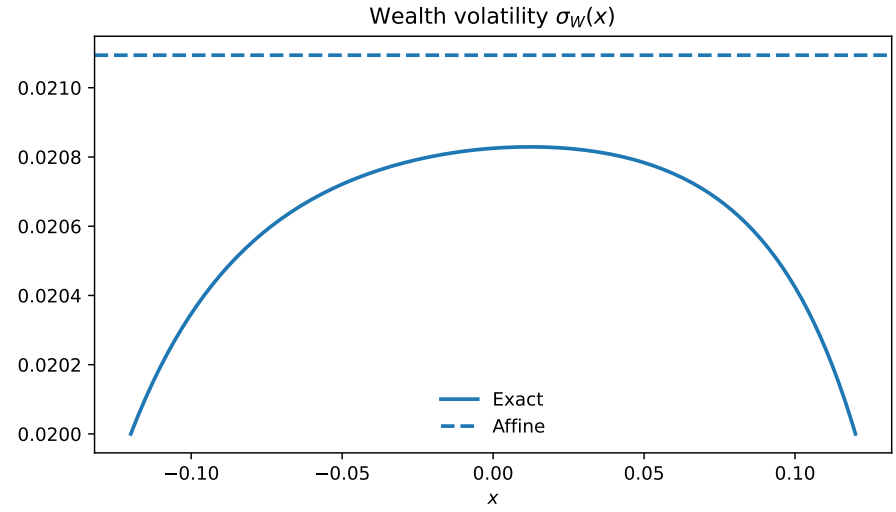



```
fig, axes = plt.subplots(3, 1, figsize=(7, 12))

axes[0].plot(x_plot, q_exact, lw=2, label="Exact")
axes[0].plot(x_plot, q_affine, "--", lw=2, label="Affine")
axes[0].set_title(r"$q(x) = \ln h(x)$")
axes[0].set_xlabel(r"$x$")
axes[0].legend(frameon=False)

axes[1].plot(x_plot, m_exact, lw=2, label="Exact")
axes[1].plot(x_plot, m_affine, "--", lw=2, label="Affine")
axes[1].set_title(r"$m(x) = c/W$")
```

```
axes[1].set_xlabel(r"$x$")
axes[1].legend(frameon=False)

axes[2].plot(x_plot, sigma_w_exact, lw=2, label="Exact")
axes[2].axhline(sigma_w_affine, ls="--", lw=2, label="Affine")
axes[2].set_title(r"Wealth volatility $\sigma_W(x)$")
axes[2].set_xlabel(r"$x$")
axes[2].legend(frameon=False)

fig.tight_layout()
plt.show()
```

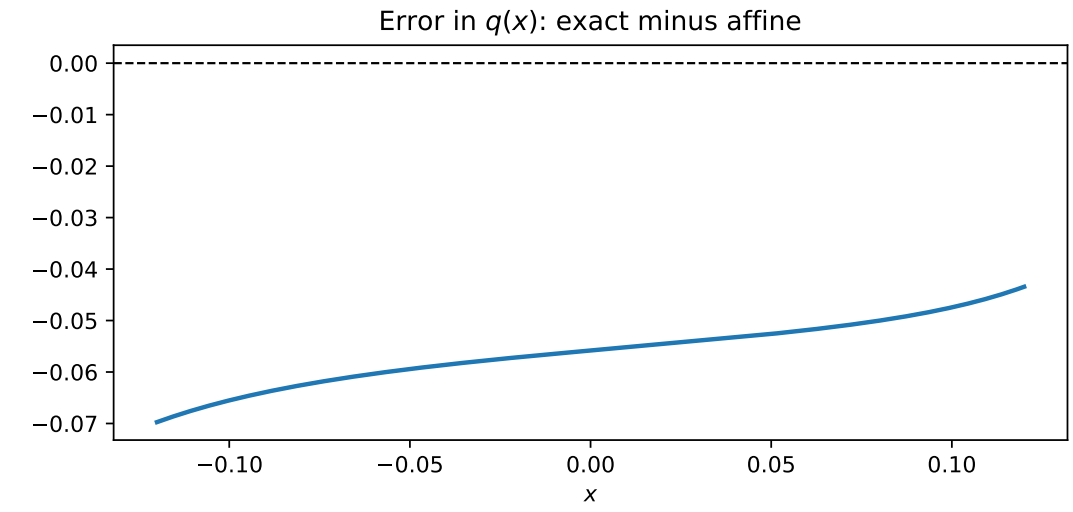

$q(x) = \ln h(x)$
Exact
Affine
23.96
23.94
23.92
23.90
23.88
23.86
−0.10
−0.05
0.00
0.05
0.10
x

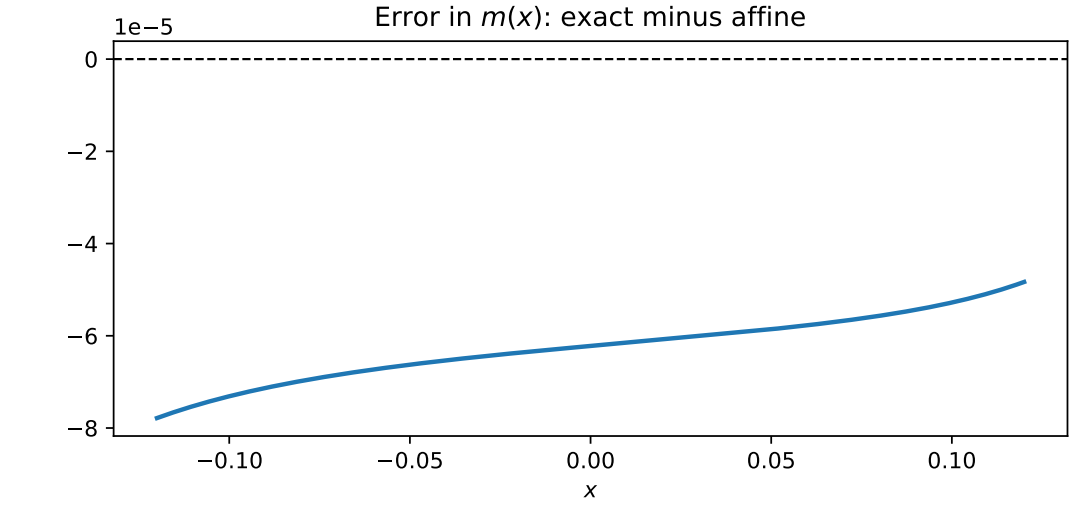

$m(x) = c/W$
Exact
Affine
0.01566
0.01564
0.01562
0.01560
0.01558
0.01556
−0.10
−0.05
0.00
0.05
0.10
x

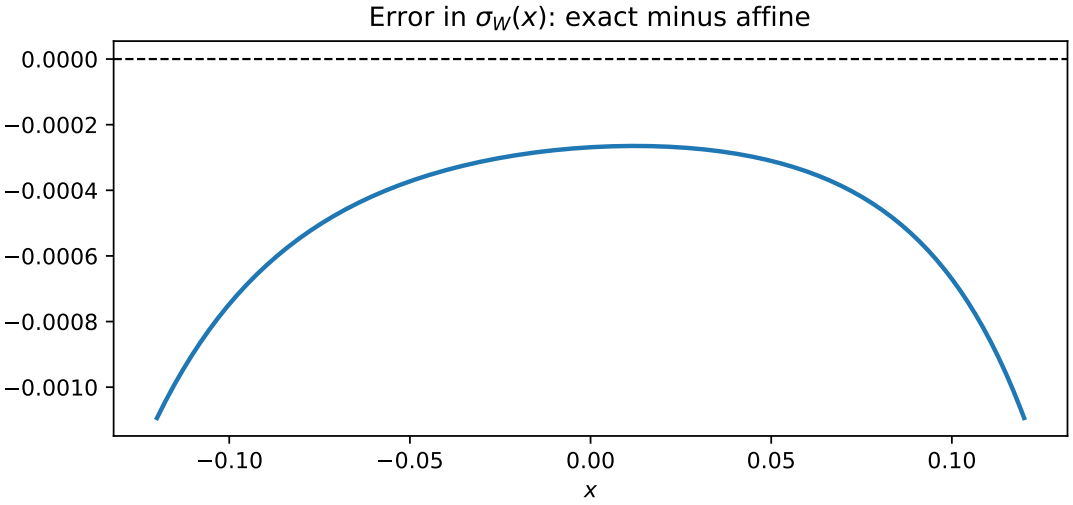

Wealth volatility $\sigma_W(x)$
Exact
Affine
0.0210
0.0208
0.0206
0.0204
0.0202
0.0200
−0.10
−0.05
0.00
0.05
0.10
x

The next figure makes the same comparison in error form by plotting exact minus affine.

```
fig, axes = plt.subplots(3, 1, figsize=(7, 10))

axes[0].plot(x_plot, q_error, lw=2)
axes[0].axhline(0.0, color="black", ls="--", lw=1)
axes[0].set_title(r"Error in $q(x)$: exact minus affine")
axes[0].set_xlabel(r"$x$")

axes[1].plot(x_plot, m_error, lw=2)
axes[1].axhline(0.0, color="black", ls="--", lw=1)
axes[1].set_title(r"Error in $m(x)$: exact minus affine")
axes[1].set_xlabel(r"$x$")

axes[2].plot(x_plot, sigma_w_error, lw=2)
axes[2].axhline(0.0, color="black", ls="--", lw=1)
axes[2].set_title(r"Error in $\sigma_W(x)$: exact minus affine")
axes[2].set_xlabel(r"$x$")

fig.tight_layout()
plt.show()
```

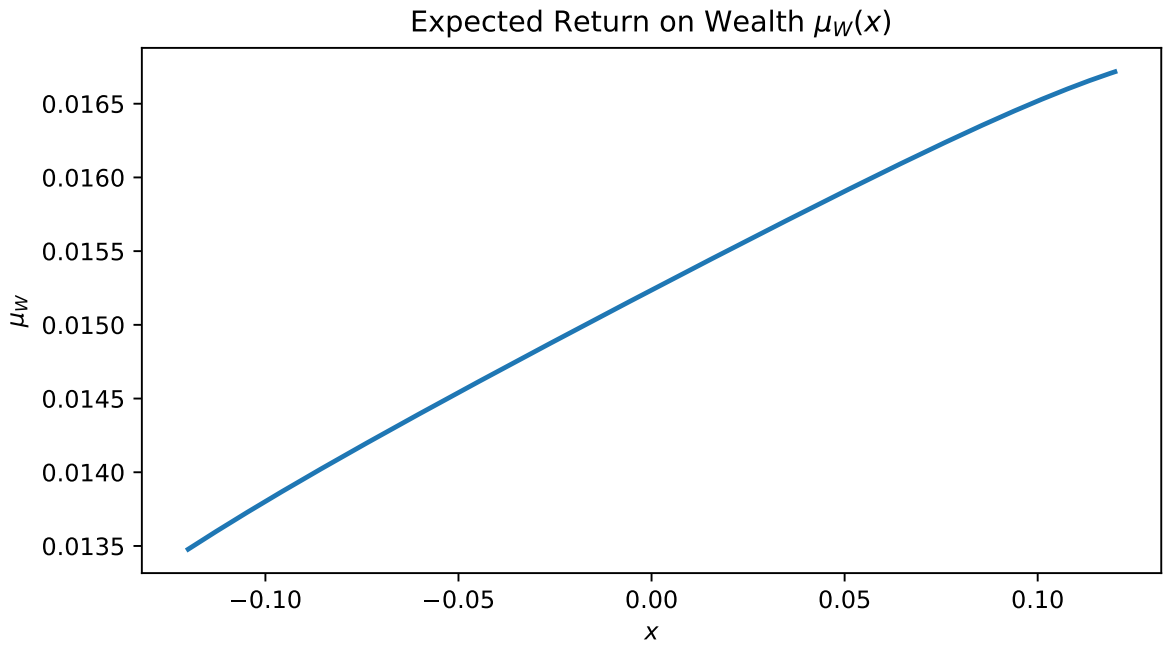

Error in $q(x)$: exact minus affine
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
−0.10
−0.05
0.00
0.05
0.10
x


Error in $m(x)$: exact minus affine
1e−5
0
−2
−4
−6
−8
−0.10
−0.05
0.00
0.05
0.10
x


Error in $\sigma_W(x)$: exact minus affine
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
−0.0010
−0.10
−0.05
0.00
0.05
0.10
x

The exact solution also delivers the endogenous expected return on aggregate wealth.

```
plt.figure(figsize=(7, 4))
plt.plot(x_plot, mu_w_exact, lw=2)
plt.title(r"Expected Return on Wealth $\mu_W(x)$")
plt.xlabel(r"$x$")
plt.ylabel(r"$\mu_W$")
plt.tight_layout()
plt.show()
```



Interpretation

Three features are worth emphasizing.

1. In this benchmark calibration, the exact and affine solutions are close for $q(x)$, $m(x)$, and $\sigma_W(x)$, so the affine approximation captures most of the quantitative variation.

2. The remaining difference is curvature: the exact solution has $q''(x) \neq 0$, while the affine approximation rules that out by construction.
3. That curvature matters most for objects such as $\mu_W(x)$ that depend directly on $q''(x)$, so the exact solution is most useful as a benchmark for what the affine approximation leaves out.

This notebook therefore complements the analytical affine solution in a focused way: the affine approximation remains the tractable workhorse, and the exact numerical value function tells us how much nonlinear curvature that workhorse is ignoring in a representative calibration.

References