In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

DT   = 1/365
ST   = 1.0
XT   = 1000.0
YT   = 1000.0
K    = XT * YT
PT   = YT / XT  # = 1.0

GAMMA_CHOICES = np.array([0.001, 0.003, 0.01])
```

In [3]:
```
def lognormal_pdf(s, sigma, dt=DT, st=ST):
    s = np.asarray(s)
    mu = np.log(st) - 0.5 * sigma**2 * dt
    v  = sigma**2 * dt
    sd = np.sqrt(v)

    z = (np.log(s) - mu) / sd
    return np.exp(-0.5 * z*z) / (s * sd * np.sqrt(2*np.pi))


def delta_y_case1(s, gamma):
    s = np.asarray(s)
    return (np.sqrt(K * (1 - gamma) * s) - YT) / (1 - gamma)

def delta_x_case2(s, gamma):
    s = np.asarray(s)
    return (np.sqrt(K * (1 - gamma) / s) - XT) / (1 - gamma)

def expected_fee_revenue(sigma, gamma, N=40000, ZMAX=10):
    upper = PT / (1 - gamma)   # PT/(1-gamma)
    lower = PT * (1 - gamma)   # PT*(1-gamma)

    mu = np.log(ST) - 0.5 * sigma**2 * DT
    sd = sigma * np.sqrt(DT)

    s_min = np.exp(mu - ZMAX * sd)
    s_max = np.exp(mu + ZMAX * sd)

    a1 = max(s_min, 1e-12)
    b1 = lower

    I1 = 0.0
    if a1 < b1:
        s1 = np.linspace(a1, b1, N)
        pdf1 = lognormal_pdf(s1, sigma)
        integrand1 = gamma * delta_x_case2(s1, gamma) * s1 * pdf1
        I1 = np.trapz(integrand1, s1)

    a2 = upper
    b2 = s_max
```

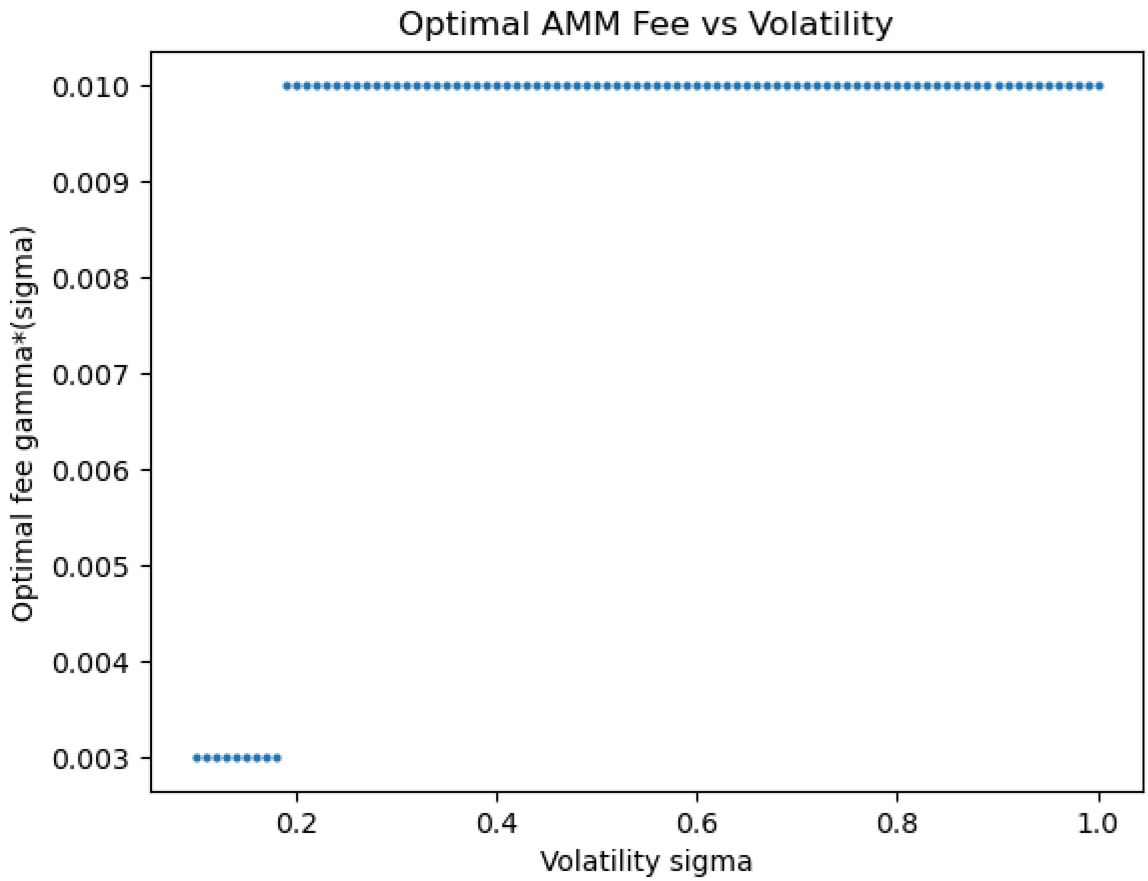
```
    I2 = 0.0
    if a2 < b2:
        s2 = np.linspace(a2, b2, N)
        pdf2 = lognormal_pdf(s2, sigma)
        integrand2 = gamma * delta_y_case1(s2, gamma) * pdf2
        I2 = np.trapz(integrand2, s2)

    return I1 + I2



sigma_b = 0.2
gamma_b = 0.003

Er_b = expected_fee_revenue(sigma_b, gamma_b, N=50000, ZMAX=10)
print("E[R(S_{t+1})] ≈", Er_b)
```

```
E[R(S_{t+1})] ≈ 0.008522036333857185
```

In [5]:

```
sigmas_c = [0.2, 0.6, 1.0]

rows = []
for s in sigmas_c:
    vals = [expected_fee_revenue(s, g, N=40000, ZMAX=10) for g in GAMMA_CHOI
    best_idx = int(np.argmax(vals))
    rows.append([s, *vals, GAMMA_CHOICES[best_idx]])

df = pd.DataFrame(
    rows,
    columns=["sigma", "E[R] (g=0.001)", "E[R] (g=0.003)", "E[R] (g=0.01)", "
)
print("\nPart (c1) table:")
print(df.to_string(index=False))


sig_grid = np.round(np.arange(0.10, 1.00 + 1e-12, 0.01), 2)
gamma_star = np.zeros_like(sig_grid, dtype=float)

# Use a slightly smaller N for the grid sweep to keep it quick
for i, s in enumerate(sig_grid):
    vals = [expected_fee_revenue(s, g, N=20000, ZMAX=10) for g in GAMMA_CHOI
    gamma_star[i] = GAMMA_CHOICES[int(np.argmax(vals))]

plt.figure()
plt.plot(sig_grid, gamma_star, marker="o", linestyle="None", markersize=2)
plt.xlabel("Volatility sigma")
plt.ylabel("Optimal fee gamma*(sigma)")
plt.title("Optimal AMM Fee vs Volatility")
plt.show()
```

```
Part (c1) table:
 sigma  E[R] (g=0.001)  E[R] (g=0.003)  E[R] (g=0.01)  gamma*(sigma)
   0.2        0.003685        0.008522       0.009430           0.01
   0.6        0.011923        0.032983       0.081082           0.01
   1.0        0.020061        0.057384       0.160690           0.01
```



In [7]:
```
import numpy as np
import math

def f1(x, y):
    return x * y

def f2(x, y):
    return math.exp(x + y)

def composite_rule(f, n, m):
    dx = 1 / n
    dy = 3 / m

    total = 0.0

    for i in range(n):
        for j in range(m):

            xi = i * dx
            xi1 = (i + 1) * dx

            yj = j * dy
            yj1 = (j + 1) * dy

            xm = (xi + xi1) / 2
            ym = (yj + yj1) / 2

            cell = (
```

```
                f(xi, yj)
                + f(xi, yj1)
                + f(xi1, yj)
                + f(xi1, yj1)

                + 2 * (f(xm, yj) + f(xm, yj1)
                       + f(xi, ym) + f(xi1, ym))

                + 4 * f(xm, ym)
            )

            total += cell

    return (dx * dy / 16) * total
```

In [11]:

```
exact_f1 = 9/4
exact_f2 = (math.exp(3) - 1) * (math.exp(1) - 1)

grids = [(2,2), (4,4), (8,8), (16,16)]

print("n  m   Approx f1   Error f1   Approx f2   Error f2")

for n, m in grids:
    approx1 = composite_rule(f1, n, m)
    approx2 = composite_rule(f2, n, m)

    error1 = abs(exact_f1 - approx1)
    error2 = abs(exact_f2 - approx2)

    print(f"{n:<3}{m:<4}{approx1:>10.6f}   {error1:>10.6f}   {approx2:>10.6f
```

```
n  m   Approx f1   Error f1   Approx f2   Error f2
2  2     2.250000    0.000000    34.495895     1.701563
4  4     2.250000    0.000000    33.220931     0.426600
8  8     2.250000    0.000000    32.901058     0.106727
16 16    2.250000    0.000000    32.821018     0.026686
```

In [ ]: