NBA5420_A1

February 22, 2026

0.0.1 (A) Three Plain Portfolio Simulations

[6]:
```
import numpy as np
import pandas as pd

# ----------------------------
# 1) Capital market assumptions (from your images)
# Order: Cash, UST Int, IG, ACWI, HF, PRE, PE
# ----------------------------
mu = np.array([0.0310, 0.0406, 0.0546, 0.0828, 0.0546, 0.0879, 0.1191])
sd = np.array([0.0067, 0.0348, 0.0739, 0.1678, 0.0575, 0.1139, 0.1978])

corr = np.array([
    [1.00, 0.20, 0.02, 0.03, 0.04, -0.12, -0.01],
    [0.20, 1.00, 0.55, 0.00, -0.27, -0.28, -0.31],
    [0.02, 0.55, 1.00, 0.56, 0.35, -0.02, 0.35],
    [0.03, 0.00, 0.56, 1.00, 0.72, 0.32, 0.81],
    [0.04, -0.27, 0.35, 0.72, 1.00, 0.33, 0.76],
    [-0.12, -0.28, -0.02, 0.32, 0.33, 1.00, 0.32],
    [-0.01, -0.31, 0.35, 0.81, 0.76, 0.32, 1.00]
], dtype=float)

cov = np.outer(sd, sd) * corr

# ----------------------------
# 2) Cash-flow inputs (from your Step 1 / Exhibit 8)
# ----------------------------
A0 = 9.12e9
c0 = 388_767_669.0     # 2016 regular contribution (excl supplement)
p0 = 816_122_544.0     # 2016 payout
g_c = 0.0261
g_p = 0.0671
addl_2017 = 353_800_000.0  # one-time injection in 2017

# ----------------------------
# 3) Portfolio weights (from your efficient frontier table)
# ----------------------------
# Order: Cash, UST Int, IG, ACWI, HF, PRE, PE
```

```
w_75 = np.array([0.0, 0.0, 0.3234, 0.3766, 0.10,   0.10, 0.10])
w_80 = np.array([0.0, 0.0, 0.1786, 0.5539, 0.0675, 0.10, 0.10])
w_85 = np.array([0.0, 0.0, 0.0688, 0.7312, 0.0,    0.10, 0.10])

def simulate_portfolio(weights, n_paths=50_000, n_years=20, seed=0):
    rng = np.random.default_rng(seed)

    # Simulate correlated asset-class returns
    R = rng.multivariate_normal(mean=mu, cov=cov, size=(n_paths, n_years))  #␣
 ↪(paths, years, 7)
    Rp = R @ weights  # portfolio return each year, shape (paths, years)

    # Build contribution/payout schedules
    contrib = np.empty(n_years)
    payout  = np.empty(n_years)

    contrib[0] = c0 * (1 + g_c) + addl_2017
    payout[0]  = p0 * (1 + g_p)

    for t in range(1, n_years):
        # grow only the regular part after 2017
        prev_reg = contrib[t-1] - (addl_2017 if t-1 == 0 else 0.0)
        contrib[t] = prev_reg * (1 + g_c)
        payout[t]  = payout[t-1] * (1 + g_p)

    # Fund evolution (returns on beginning assets; flows at year-end)
    A = np.full(n_paths, A0, dtype=float)
    ruined = np.zeros(n_paths, dtype=bool)

    for t in range(n_years):
        A = A * (1 + Rp[:, t]) + contrib[t] - payout[t]
        ruined |= (A <= 0)
        A = np.where(ruined, 0.0, A)

    prob_ruin = ruined.mean()
    avg_final = A.mean()
    return prob_ruin, avg_final

portfolios = {
    "7.5% (SD 10.29%)": w_75,
    "8.0% (SD 12.41%)": w_80,
    "8.5% (SD 14.61%)": w_85,
}

rows = []
for name, w in portfolios.items():
    prob, avg = simulate_portfolio(w, seed=0)
```

```
    rows.append([name, prob, avg, avg/1e9])

df = pd.DataFrame(rows, columns=["Portfolio", "P(Ruin by 20y)", "Avg Final␣
 ↪($)", "Avg Final (B$)"])
print(df.to_string(index=False))
```

```
      Portfolio  P(Ruin by 20y)  Avg Final ($)  Avg Final (B$)
7.5% (SD 10.29%)         0.69108   2.969672e+09        2.969672
8.0% (SD 12.41%)         0.63210   5.038389e+09        5.038389
8.5% (SD 14.61%)         0.59208   7.568046e+09        7.568046
```

Q(c)

[7]:
```
import numpy as np
import pandas as pd

# ----------------------------
# Capital market assumptions
# ----------------------------
mu = np.array([0.0310, 0.0406, 0.0546, 0.0828, 0.0546, 0.0879, 0.1191])
sd = np.array([0.0067, 0.0348, 0.0739, 0.1678, 0.0575, 0.1139, 0.1978])

corr = np.array([
    [1.00, 0.20, 0.02, 0.03, 0.04, -0.12, -0.01],
    [0.20, 1.00, 0.55, 0.00, -0.27, -0.28, -0.31],
    [0.02, 0.55, 1.00, 0.56, 0.35, -0.02, 0.35],
    [0.03, 0.00, 0.56, 1.00, 0.72, 0.32, 0.81],
    [0.04, -0.27, 0.35, 0.72, 1.00, 0.33, 0.76],
    [-0.12, -0.28, -0.02, 0.32, 0.33, 1.00, 0.32],
    [-0.01, -0.31, 0.35, 0.81, 0.76, 0.32, 1.00]
])

cov = np.outer(sd, sd) * corr

# ----------------------------
# Cash flow assumptions
# ----------------------------
A0 = 9.12e9
c0 = 388_767_669.0
p0 = 816_122_544.0
g_c = 0.0261
g_p = 0.0671

supplement_2017 = 353_800_000.0
supplement_years = 10    # 2017-2026 inclusive

# ----------------------------
# Portfolio weights
```

```python
# ----------------------------
w_75 = np.array([0,0,0.3234,0.3766,0.10,0.10,0.10])
w_80 = np.array([0,0,0.1786,0.5539,0.0675,0.10,0.10])
w_85 = np.array([0,0,0.0688,0.7312,0.0,0.10,0.10])

def simulate_portfolio(weights, n_paths=50000, n_years=20, seed=1):
    rng = np.random.default_rng(seed)
    R = rng.multivariate_normal(mu, cov, size=(n_paths, n_years))
    Rp = R @ weights

    A = np.full(n_paths, A0)
    ruined = np.zeros(n_paths, dtype=bool)

    regular_contrib = c0
    supplement = supplement_2017
    payout = p0

    for t in range(n_years):

        # grow regular contribution
        regular_contrib *= (1 + g_c)

        # grow payout
        payout *= (1 + g_p)

        # grow supplement only for first 10 years
        if t < supplement_years:
            supplement *= (1 + g_c)
            total_contrib = regular_contrib + supplement
        else:
            total_contrib = regular_contrib

        # evolve assets
        A = A * (1 + Rp[:, t]) + total_contrib - payout

        ruined |= (A <= 0)
        A = np.where(ruined, 0, A)

    return ruined.mean(), A.mean()

portfolios = {
    "7.5%": w_75,
    "8.0%": w_80,
    "8.5%": w_85
}

results = []
```

```
for name, w in portfolios.items():
    prob, avg = simulate_portfolio(w)
    results.append([name, prob, avg])

df = pd.DataFrame(results, columns=["Portfolio", "Prob Ruin", "Avg Final␣
 ↪Assets"])
print(df)
```

```
  Portfolio  Prob Ruin  Avg Final Assets
0      7.5%    0.32802       8.896786e+09
1      8.0%    0.32486       1.190133e+10
2      8.5%    0.33058       1.527369e+10
```

Q(b)

[8]:
```
import numpy as np
import pandas as pd

# ----- CMA inputs (order: Cash, UST Int, IG, ACWI, HF, PRE, PE)
mu = np.array([0.0310, 0.0406, 0.0546, 0.0828, 0.0546, 0.0879, 0.1191])
sd = np.array([0.0067, 0.0348, 0.0739, 0.1678, 0.0575, 0.1139, 0.1978])

corr = np.array([
    [1.00, 0.20, 0.02, 0.03, 0.04, -0.12, -0.01],
    [0.20, 1.00, 0.55, 0.00, -0.27, -0.28, -0.31],
    [0.02, 0.55, 1.00, 0.56, 0.35, -0.02, 0.35],
    [0.03, 0.00, 0.56, 1.00, 0.72, 0.32, 0.81],
    [0.04, -0.27, 0.35, 0.72, 1.00, 0.33, 0.76],
    [-0.12, -0.28, -0.02, 0.32, 0.33, 1.00, 0.32],
    [-0.01, -0.31, 0.35, 0.81, 0.76, 0.32, 1.00]
], dtype=float)

cov = np.outer(sd, sd) * corr

# ----- Cash-flow inputs
A0 = 9.12e9
c0 = 388_767_669.0       # 2016 regular contribution excl. supplement
p0 = 816_122_544.0       # 2016 payout
g_c = 0.0261
g_p = 0.0671

# ----- Frontier weights
w_75 = np.array([0.0, 0.0, 0.3234, 0.3766, 0.10,   0.10, 0.10])
w_80 = np.array([0.0, 0.0, 0.1786, 0.5539, 0.0675, 0.10, 0.10])
w_85 = np.array([0.0, 0.0, 0.0688, 0.7312, 0.0,    0.10, 0.10])

def simulate_with_20y_supp(weights, S2017, g_s=None, n_paths=50_000,␣
 ↪n_years=20, seed=0):
```

```
    """
    Supplement runs for all 20 years: S_t = S2017*(1+g_s)^(t) for t=0..19 (2017.
↪.2036).
    If g_s is None, default to g_c.
    """
    if g_s is None:
        g_s = g_c

    rng = np.random.default_rng(seed)
    R = rng.multivariate_normal(mu, cov, size=(n_paths, n_years))
    Rp = R @ weights  # portfolio returns per year

    # Build regular contribution / payout schedules for 2017..2036
    reg = np.empty(n_years)
    pay = np.empty(n_years)
    supp = np.empty(n_years)

    reg[0] = c0 * (1 + g_c)
    pay[0] = p0 * (1 + g_p)
    supp[0] = S2017

    for t in range(1, n_years):
        reg[t]  = reg[t-1] * (1 + g_c)
        pay[t]  = pay[t-1] * (1 + g_p)
        supp[t] = supp[t-1] * (1 + g_s)

    contrib = reg + supp

    A = np.full(n_paths, A0, dtype=float)
    ruined = np.zeros(n_paths, dtype=bool)

    for t in range(n_years):
        A = A * (1 + Rp[:, t]) + contrib[t] - pay[t]
        ruined |= (A <= 0)
        A = np.where(ruined, 0.0, A)

    return ruined.mean(), A.mean()

def calibrate_S2017(weights, target_ruin=0.10, g_s=None, seed=0,
                    lo=0.0, hi=5e9, tol=1e-4, max_iter=40):
    """
    Finds smallest S2017 such that ruin_prob <= target_ruin (bisection).
    hi default 5B; increase if needed.
    """
    # Ensure hi is large enough
    pr_hi, _ = simulate_with_20y_supp(weights, hi, g_s=g_s, seed=seed)
    while pr_hi > target_ruin:
```

```
        hi *= 2
        if hi > 1e11:
            raise RuntimeError("Upper bound exploded; check assumptions.")
        pr_hi, _ = simulate_with_20y_supp(weights, hi, g_s=g_s, seed=seed)

    pr_lo, _ = simulate_with_20y_supp(weights, lo, g_s=g_s, seed=seed)
    if pr_lo <= target_ruin:
        return lo, pr_lo

    for _ in range(max_iter):
        mid = 0.5 * (lo + hi)
        pr_mid, avg_mid = simulate_with_20y_supp(weights, mid, g_s=g_s,␣
 ↪seed=seed)
        if pr_mid <= target_ruin:
            hi = mid
        else:
            lo = mid
        if abs(pr_mid - target_ruin) < tol:
            break

    # final at hi (smallest meeting condition)
    pr_final, avg_final = simulate_with_20y_supp(weights, hi, g_s=g_s,␣
 ↪seed=seed)
    return hi, pr_final, avg_final

portfolios = [
    ("Baseline Portfolio", 0.075, 0.1029, w_75),
    ("Moderate Growth Portfolio", 0.080, 0.1241, w_80),
    ("Growth-Oriented Portfolio", 0.085, 0.1461, w_85),
]

# Example run: fix g_s = g_c and target ruin = 10%
rows = []
for name, er, sd_p, w in portfolios:
    S_star, pr_star, avg_star = calibrate_S2017(w, target_ruin=0.10, g_s=g_c,␣
 ↪seed=0)
    rows.append([name, er, sd_p, S_star, g_c, pr_star, avg_star])

df = pd.DataFrame(rows, columns=[
    "Portfolio", "E[R]", "SD", "Required S_2017", "g_s", "Ruin Prob (20y)",␣
 ↪"Avg Final Assets"
])
print(df.to_string(index=False))
```

```
                Portfolio  E[R]     SD  Required S_2017    g_s  Ruin Prob (20y)
Avg Final Assets
       Baseline Portfolio 0.075 0.1029     3.564453e+08 0.0261          0.09936
```

```
1.455976e+10
Moderate Growth Portfolio 0.080 0.1241     3.833008e+08 0.0261          0.09942
1.898279e+10
Growth-Oriented Portfolio 0.085 0.1461     4.147339e+08 0.0261          0.09994
2.413442e+10
```

0.1 COLA scenario analysis

[12]:
```python
# Q4(a) parameters (Unchanged)
A0 = 9.12e9
c0 = 388_767_669.0
p0 = 816_122_544.0
g_c = 0.0261
addl_2017 = 353_800_000.0

# payout growth decomposition
g_total = 0.0671
cola_base = 0.025
g_demo = g_total - cola_base   # 0.0421

def simulate_portfolio_q5(weights, cola=0.025, n_paths=50_000, n_years=20,␣
 ↪seed=0):
    """
    Q5 on top of Q4(a):
    - one-time injection in 2017 only (to fully examine the impact brought by␣
 ↪COLA)
    - contributions grow at g_c
    - payout growth = g_demo + cola
    """
    rng = np.random.default_rng(seed)

    # 1) Generate and weight the relevant asset returns
    R = rng.multivariate_normal(mean=mu, cov=cov, size=(n_paths, n_years))
    Rp = R @ weights

    # 2) this scenario's payout growth
    g_p = g_demo + cola

    # 3) build schedule 2017..2036
    contrib = np.empty(n_years)
    payout  = np.empty(n_years)

    # 2017: Regular contribution growth + one-time funding; payout grows␣
 ↪according to the new g_p
    contrib[0] = c0 * (1 + g_c) + addl_2017
    payout[0]  = p0 * (1 + g_p)
```

```
    for t in range(1, n_years):
        prev_reg = contrib[t-1] - (addl_2017 if t-1 == 0 else 0.0)  #
        contrib[t] = prev_reg * (1 + g_c)
        payout[t]  = payout[t-1] * (1 + g_p)

    # 4) Asset Evolution + Ruin Rules
    A = np.full(n_paths, A0, dtype=float)
    ruined = np.zeros(n_paths, dtype=bool)

    for t in range(n_years):
        A = A * (1 + Rp[:, t]) + contrib[t] - payout[t]
        ruined |= (A <= 0)
        A = np.where(ruined, 0.0, A)

    return ruined.mean(), A.mean()
```

Sanity check First, run it once with COLA = 2.5% to see if we can reproduce the results of your Q4(a) (a little Monte Carlo noise is acceptable).

[11]:
```
portfolios = {
    "Baseline (7.5%)": w_75,
    "Moderate (8.0%)": w_80,
    "Growth (8.5%)": w_85
}
```

[13]:
```
for name, w in portfolios.items():
    pr, avg = simulate_portfolio_q5(w, cola=0.025, seed=0)
    print(name, pr, avg/1e9)
```

```
Baseline (7.5%) 0.69108 2.969671922419326
Moderate (8.0%) 0.6321 5.038388865220498
Growth (8.5%) 0.59208 7.5680464591874195
```

[14]:
```
colas = [0.00, 0.01, 0.015, 0.02, 0.025, 0.03]

rows = []
for cola in colas:
    for pname, w in portfolios.items():
        pr, avg = simulate_portfolio_q5(w, cola=cola, seed=0)
        rows.append([cola, 0.0421 + cola, pname, pr, avg/1e9])

df = pd.DataFrame(rows, columns=["COLA", "g_payout", "Portfolio", "P(Ruin by␣
↪20y)", "Avg Final (B$)"])
df
```

[14]:

	COLA	g_payout	Portfolio	P(Ruin by 20y)	Avg Final (B$)
0	0.000	0.0421	Baseline (7.5%)	0.17016	10.851033
1	0.000	0.0421	Moderate (8.0%)	0.19334	13.525998

2	0.000	0.0421	Growth (8.5%)	0.21704	16.525249
3	0.010	0.0521	Baseline (7.5%)	0.36348	7.135044
4	0.010	0.0521	Moderate (8.0%)	0.35606	9.709791
5	0.010	0.0521	Growth (8.5%)	0.35872	12.624257
6	0.015	0.0571	Baseline (7.5%)	0.47592	5.517299
7	0.015	0.0571	Moderate (8.0%)	0.45022	7.977090
8	0.015	0.0571	Growth (8.5%)	0.43670	10.804099
9	0.020	0.0621	Baseline (7.5%)	0.58838	4.121269
10	0.020	0.0621	Moderate (8.0%)	0.54256	6.412341
11	0.020	0.0621	Growth (8.5%)	0.51600	9.112611
12	0.025	0.0671	Baseline (7.5%)	0.69108	2.969672
13	0.025	0.0671	Moderate (8.0%)	0.63210	5.038389
14	0.025	0.0671	Growth (8.5%)	0.59208	7.568046
15	0.030	0.0721	Baseline (7.5%)	0.77930	2.063431
16	0.030	0.0721	Moderate (8.0%)	0.71188	3.868727
17	0.030	0.0721	Growth (8.5%)	0.66352	6.190329

[15]:
```
pivot_ruin = df.pivot(index="COLA", columns="Portfolio", values="P(Ruin by␣
↪20y)")
pivot_ruin
```

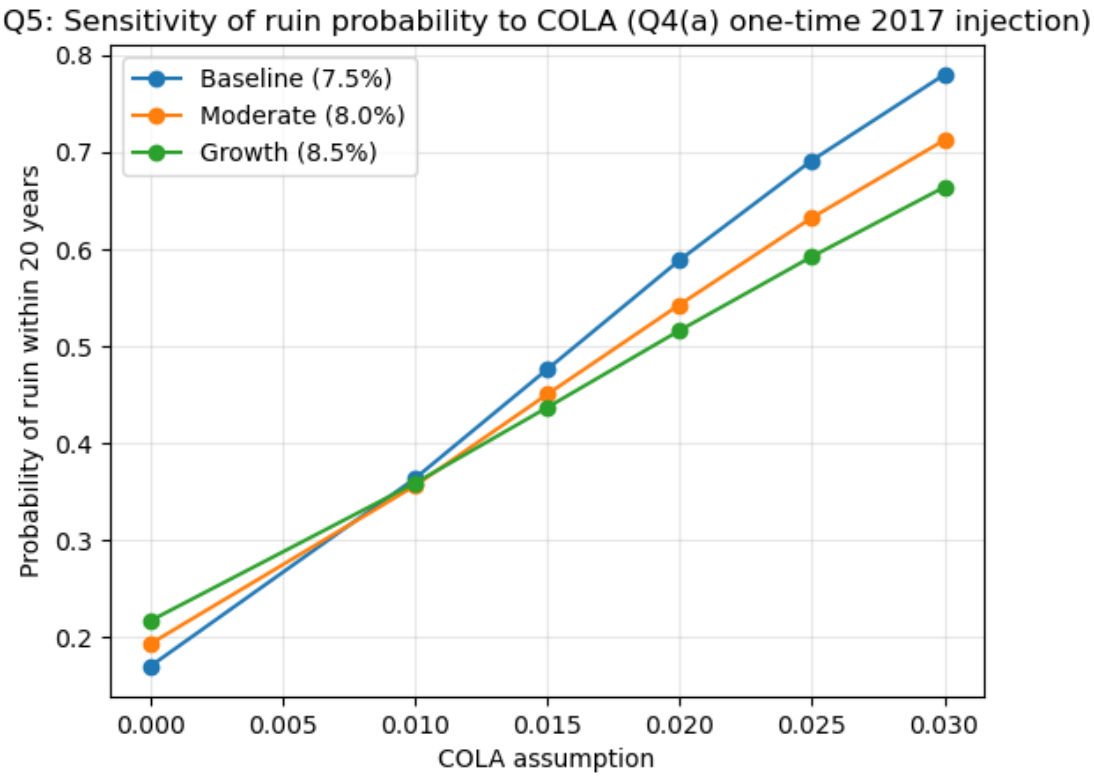
[15]:

COLA	Baseline (7.5%)	Growth (8.5%)	Moderate (8.0%)
0.000	0.17016	0.21704	0.19334
0.010	0.36348	0.35872	0.35606
0.015	0.47592	0.43670	0.45022
0.020	0.58838	0.51600	0.54256
0.025	0.69108	0.59208	0.63210
0.030	0.77930	0.66352	0.71188

[16]:
```
import matplotlib.pyplot as plt

for pname in portfolios.keys():
    s = df[df["Portfolio"] == pname]
    plt.plot(s["COLA"], s["P(Ruin by 20y)"], marker="o", label=pname)

plt.xlabel("COLA assumption")
plt.ylabel("Probability of ruin within 20 years")
plt.title("Q5: Sensitivity of ruin probability to COLA (Q4(a) one-time 2017␣
↪injection)")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```


Q5: Sensitivity of ruin probability to COLA (Q4(a) one-time 2017 injection)
Baseline (7.5%)
Moderate (8.0%)
Growth (8.5%)
Probability of ruin within 20 years
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.000
0.005
0.010
0.015
0.020
0.025
0.030
COLA assumption