Case Assignment 1

-- adapted by Thomas Cassidey from *Practical Management Science*

Base Case

James Judson is the financial manager in charge of the company pension fund at Armco Incorporated. James knows that the fund must be sufficient to make the payments listed in Table 4.4. Each payment must be made on the first day of each year. James is going to finance these payments by purchasing bonds. It is currently January 1 of year 1, and three bonds are available for immediate purchase. The prices and coupons for the bonds are as follows. (We assume that all coupon payments are received on January 1 and arrive in time to meet cash demands for the year on which they arrive.)

- Bond 1 costs $980 and yields a 60 coupon in the years 2 through 5 and a $1060 payment on maturity in the year 6.
- Bond 2 costs $970 and yields a 65 coupon in the years 2 through 11 and a $1065 payment on maturity in the year 12.
- Bond 3 costs $1050 and yields a 75 coupon in the years 2 through 14 and a $1075 payment on maturity in the year 15.

James must decide how much cash to allocate (from company coffers) to meet the initial $11,000 payment and buy enough bonds to make future payments. He knows that any excess cash on hand can earn an annual rate of 4% in a fixed-rate account. How should he proceed?

Develop an LP model that relates initial allocation of money and bond purchases to future cash availabilities, and to minimize the initialize allocation of money required to meet all future pension fund payments.

In [36]:
```
import gurobipy as gp
from gurobipy import GRB
from itertools import product
import numpy as np

def get_solution(m):
    print("Solution Values:")
    for var in m.getVars():
        print(f"    {var.VarName}: {var.X}")

def get_shadow(m):
    print(f"Shadow Price Information:")
    for constraint in m.getConstrs():
        print(f"    {constraint.ConstrName}: {constraint.Pi}, (RHS = 

def get_rc(m):
    print(f"Reduced Cost Information")
    for var in m.getVars():
        print(f"    {var.VarName}: {var.RC}, (Coef LB = {round(var.SA(
```

Parameters

In [47]:
```
T = 15 # length of the time horizon
B = [0, 1, 2] # bonds
COSTS = [980, 970, 1050] # costs in year one
D = [6, 12, 15] # duration (year of payout)

BASE_INCOMES = [60, 65, 75] # incomes in years 2 through (D_B - 1)
INCOMES = np.zeros((len(B), T))
PAYOUTS = [1060, 1065, 1075]
for b in B:
    for t in range(1, D[b] - 1):
        INCOMES[b, t] = BASE_INCOMES[b]
    INCOMES[b, D[b] - 1] = PAYOUTS[b]
    print(f"Bond {b} Incomes: {[x for x in INCOMES[b]]}")

BASE_r = .04

# Base Case
BASE_PAYMENTS = [x*1000 for x in [11, 12, 14, 15, 16, 18, 20, 21, 22,
BASE_CASE = {
    'r': [BASE_r] * T, # interest rates are level
    'c' : BASE_PAYMENTS # payments go up steadily
}

# Bad Case
BAD_PAYMENTS = [11, 12, 14, 20, 20, 22, 23, 24, 26, 30, 31, 31, 31, 3
BAD_CASE = {
    'r': [.04, .04, .04, .03, .01, .0, .0, .0, .0, .0, .0, .01, .02, .
    'c': [x*1000 for x in BAD_PAYMENTS] # payments increase dramatica
}
```

```
Bond 0 Incomes: [0.0, 60.0, 60.0, 60.0, 60.0, 1060.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Bond 1 Incomes: [0.0, 65.0, 65.0, 65.0, 65.0, 65.0, 65.0, 65.0, 65.
0, 65.0, 65.0, 1065.0, 0.0, 0.0, 0.0]
Bond 2 Incomes: [0.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.0, 75.
0, 75.0, 75.0, 75.0, 75.0, 75.0, 1075.0]
```

Base Case Model

In [51]:
```
payments = BASE_CASE['c']

# Create a new model
m = gp.Model('base_case')

# Create variables
I = m.addVars(B, vtype=GRB.CONTINUOUS, name="I", lb=0.0) # Investment
C = m.addVar(vtype=GRB.CONTINUOUS, name="C", lb=0.0) # Cash needed up
A = m.addVars(range(T), vtype=GRB.CONTINUOUS, name="A", lb=0.0) # Cash


# Set up constraint for Year 1's inittial cash allocation
# This constraint ensures that the cash available at the beginning of
# Initialize upfront cash requirement
A = m.addVars(range(T), vtype=GRB.CONTINUOUS, name="A", lb=0.0)  # Cas

# Define constraints for the initial cash allocation (Year 1)
# The available cash at the beginning of Year 1 (A[0]) is calculated a
# minus the total expenditure on bonds. The investment in each bond is
# times its respective cost (COSTS[b]).
m.addConstr(
    (
        A[0] == C - gp.quicksum(I[b] * COSTS[b] for b in B)
    ),
    name='Initial Cash Allocation Constraint'  # Labeling constraint i
)

# Define constraints for cash availability from Year 2 onward
# In each period t (from Year 2 to T), the available cash (A[t]) is ca
# reduced by pension payments (payments[t-1]), and increased by the in
# Income from bond investments in each period is added, calculated by
m.addConstrs(
    (
        A[t] == (A[t-1] - payments[t-1]) * (1 + BASE_r) + gp.quicksum(
        for t in range(1, T)  # Iterating over each year starting from
    ),
    name="Cash Flow Constraints per Period"  # Labeling constraints fo
)

# Define constraints to ensure available cash always meets or exceeds
# This constraint checks that in each period, available cash (A[t]) is
m.addConstrs(
    (
        A[t] >= payments[t] for t in range(T)  # Ensuring sufficient c
    ),
    name='Pension Payment Coverage'  # Labeling constraints to verify
)

# Objective function: Minimize the initial cash allocation (C)
# The objective is to keep the initial cash allocation (C) as low as p
m.setObjective(
    C,
    GRB.MINIMIZE  # Objective set to minimize initial cash required
)

# Run the optimization process
m.optimize()

# Display the optimal objective value (minimum initial cash required)
# The optimal value of C (minimum cash needed) is accessed with m.objV
print(f"Optimal objective value: ${round(m.objVal, 2)}")
```

```
# Extract and display the solution values (investment decisions, avail
get_solution(m)

# Obtain and display the shadow prices (dual values) for each constrai
# Shadow prices provide insight into how objective value changes with
get_shadow(m)

# Retrieve and display reduced costs for the decision variables
# Reduced costs show potential improvement in objective if non-basic v
get_rc(m)
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[rosetta2] -
Darwin 24.0.0 24A348)

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to
10 threads

Optimize a model with 30 rows, 34 columns and 78 nonzeros
Model fingerprint: 0x2bf82b8d
Coefficient statistics:
  Matrix range     [1e+00, 1e+03]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+04, 3e+04]
Presolve removed 16 rows and 17 columns
Presolve time: 0.00s
Presolved: 14 rows, 17 columns, 57 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.1000000e+04   3.459274e+04   0.000000e+00      0s
       9    1.9776840e+05   0.000000e+00   0.000000e+00      0s

Solved in 9 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.977683968e+05
Optimal objective value: $197768.4
Solution Values:
    I[0]: 73.69479810424208
    I[1]: 77.20837201549739
    I[2]: 28.837209302325583
    C: 197768.39680049522
    A[0]: 0.0
    A[1]: 0.0
    A[2]: 0.0
    A[3]: 0.0
    A[4]: 0.0
    A[5]: 0.0
    A[6]: 0.0
    A[7]: 0.0
    A[8]: 0.0
    A[9]: 0.0
    A[10]: 0.0
    A[11]: 0.0
    A[12]: 0.0
    A[13]: 0.0
    A[14]: 0.0
    A[0]: 20376.304035863654
    A[1]: 21354.378962234478
    A[2]: 21331.57688566013
    A[3]: 19227.862726022817
    A[4]: 16000.0
    A[5]: 85297.82086917835
    A[6]: 77171.06858262724
    A[7]: 66639.2462046141
    A[8]: 54646.15093148042
    A[9]: 41133.33184742139
    A[10]: 25000.0
    A[11]: 84389.70689417917
    A[12]: 58728.08586762075
    A[13]: 31000.0
    A[14]: 31000.0
Shadow Price Information:
```

```
    Initial Cash Allocation Constraint: -1.0, (RHS = 0.0), (LB = -in
f), (UB = 197768.4)
    Cash Flow Constraints per Period[1]: -0.9615384615384615, (RHS =
-11440.0), (LB = -inf), (UB = -1688.64)
    Cash Flow Constraints per Period[2]: -0.9245562130177513, (RHS =
-12480.0), (LB = -inf), (UB = -2751.45)
    Cash Flow Constraints per Period[3]: -0.8889963586709146, (RHS =
-14560.0), (LB = -inf), (UB = -6935.16)
    Cash Flow Constraints per Period[4]: -0.8548041910297255, (RHS =
-15600.0), (LB = -inf), (UB = -11203.02)
    Cash Flow Constraints per Period[5]: -0.7190625344760272, (RHS =
-16640.0), (LB = -62274.39), (UB = 61476.49)
    Cash Flow Constraints per Period[6]: -0.6914062831500261, (RHS =
-18720.0), (LB = -66179.76), (UB = 51269.73)
    Cash Flow Constraints per Period[7]: -0.6648137337981019, (RHS =
-20800.0), (LB = -70158.15), (UB = 38657.91)
    Cash Flow Constraints per Period[8]: -0.6392439748058671, (RHS =
-21840.0), (LB = -73172.48), (UB = 25624.82)
    Cash Flow Constraints per Period[9]: -0.6146576680825645, (RHS =
-22880.0), (LB = -76265.78), (UB = 11072.0)
    Cash Flow Constraints per Period[10]: -0.5910169885409273, (RHS
= -24960.0), (LB = -80481.21), (UB = -7141.33)
    Cash Flow Constraints per Period[11]: -0.44999411599795885, (RHS
= -26000.0), (LB = -87212.56), (UB = 56226.92)
    Cash Flow Constraints per Period[12]: -0.4326866499980373, (RHS
= -31200.0), (LB = -94861.06), (UB = 25365.3)
    Cash Flow Constraints per Period[13]: -0.4160448557673435, (RHS
= -32240.0), (LB = -98447.5), (UB = -3402.79)
    Cash Flow Constraints per Period[14]: -0.3593845096134623, (RHS
= -32240.0), (LB = -97238.23), (UB = -1240.0)
    Pension Payment Coverage[0]: 0.0, (RHS = 11000.0), (LB = -inf),
(UB = 20376.3)
    Pension Payment Coverage[1]: 0.0, (RHS = 12000.0), (LB = -inf),
(UB = 21354.38)
    Pension Payment Coverage[2]: 0.0, (RHS = 14000.0), (LB = -inf),
(UB = 21331.58)
    Pension Payment Coverage[3]: 0.0, (RHS = 15000.0), (LB = -inf),
(UB = 19227.86)
    Pension Payment Coverage[4]: 0.10697915517465717, (RHS = 16000.
0), (LB = 11847.47), (UB = 91112.01)
    Pension Payment Coverage[5]: 0.0, (RHS = 18000.0), (LB = -inf),
(UB = 85297.82)
    Pension Payment Coverage[6]: 0.0, (RHS = 20000.0), (LB = -inf),
(UB = 77171.07)
    Pension Payment Coverage[7]: 0.0, (RHS = 21000.0), (LB = -inf),
(UB = 66639.25)
    Pension Payment Coverage[8]: 0.0, (RHS = 22000.0), (LB = -inf),
(UB = 54646.15)
    Pension Payment Coverage[9]: 0.0, (RHS = 24000.0), (LB = -inf),
(UB = 41133.33)
    Pension Payment Coverage[10]: 0.1230231079030501, (RHS = 25000.
0), (LB = 8244.85), (UB = 104064.34)
    Pension Payment Coverage[11]: 0.0, (RHS = 30000.0), (LB = -inf),
(UB = 84389.71)
    Pension Payment Coverage[12]: 0.0, (RHS = 31000.0), (LB = -inf),
(UB = 58728.09)
    Pension Payment Coverage[13]: 0.04228496576934271, (RHS = 31000.
0), (LB = 4113.62), (UB = 60807.69)
    Pension Payment Coverage[14]: 0.3593845096134623, (RHS = 31000.
0), (LB = -0.0), (UB = 95998.23)
Reduced Cost Information
```

```
I[0]: 0.0, (Coef LB = -111.65), (Coef UB = 109.04)
I[1]: 0.0, (Coef LB = -39.71), (Coef UB = 125.98)
I[2]: 0.0, (Coef LB = -386.34), (Coef UB = 43.71)
C: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[0]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[1]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[2]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[3]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[4]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[5]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[6]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[7]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[8]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[9]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[10]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[11]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[12]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[13]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[14]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
A[0]: 0.0, (Coef LB = -0.1), (Coef UB = 2.49)
A[1]: 0.0, (Coef LB = -0.1), (Coef UB = 3.26)
A[2]: 0.0, (Coef LB = -0.1), (Coef UB = 4.79)
A[3]: 0.0, (Coef LB = -0.11), (Coef UB = 9.4)
A[4]: 0.0, (Coef LB = -0.11), (Coef UB = inf)
A[5]: 0.0, (Coef LB = -0.11), (Coef UB = 1.4)
A[6]: 0.0, (Coef LB = -0.11), (Coef UB = 1.72)
A[7]: 0.0, (Coef LB = -0.12), (Coef UB = 2.25)
A[8]: 0.0, (Coef LB = -0.12), (Coef UB = 3.32)
A[9]: 0.0, (Coef LB = -0.12), (Coef UB = 6.5)
A[10]: 0.0, (Coef LB = -0.12), (Coef UB = inf)
A[11]: 0.0, (Coef LB = -0.04), (Coef UB = 2.73)
A[12]: 0.0, (Coef LB = -0.04), (Coef UB = 5.36)
A[13]: 0.0, (Coef LB = -0.04), (Coef UB = inf)
A[14]: 0.0, (Coef LB = -0.36), (Coef UB = inf)
```

Bad Case

The model above assumes a steady interest rate for all time periods $t = 1, \ldots, T$, but that is probably not a safe assumption. For example, an economic downturn is often accompanied by a dropping of interest rates. Further, this may be accompanied by an icrease in payments to the pension fund due to early retirements, etc. Develop a LP model in the cell below that uses the interest rates and payments of the `BAD_CASE` shown above. Assume that the interest rate for cash carried from year t to year t+1 is the interest rate for year t.

Bad Case Model

In [53]:
```
# Extract pension payments and interest rates from the BAD_CASE scenar
payments = BAD_CASE['c'] # pension payments required in the bad case
interest_rates = BAD_CASE['r'] #Annual interest rates specific to this

# Create a new model
m = gp.Model('base_case')

# Create variables
I = m.addVars(B, vtype=GRB.CONTINUOUS, name="I", lb=0.0) # Investment
C = m.addVar(vtype=GRB.CONTINUOUS, name="C", lb=0.0) # Cash needed up
A = m.addVars(range(T), vtype=GRB.CONTINUOUS, name="A", lb=0.0) # Cash

# Define decision variables

# I[b]: How much to invest in each bond (number of units to buy for ea
# These variables are continuous and non-negative (lb=0.0).
I = m.addVars(B, vtype=GRB.CONTINUOUS, name="I", lb=0.0)

# C: Initial cash needed upfront to cover Year 1 pension payments and
# This variable is continuous and non-negative.
C = m.addVar(vtype=GRB.CONTINUOUS, name="C", lb=0.0)

# A[t]: Cash available at the beginning of each year t (from Year 1 to
# These variables are continuous and non-negative.
A = m.addVars(range(T), vtype=GRB.CONTINUOUS, name="A", lb=0.0)

# Set up constraints

# Year 1 constraint: Ensure that cash available at the start of Year 1
# minus the total cost of buying bonds. The cost is calculated by multi
m.addConstr(
    (
        A[0] == C - gp.quicksum(I[b] * COSTS[b] for b in B)
    ),
    name='Initial Cash Constraint'  # Label for Year 1 cash availabili
)

# Constraints for Years 2 through T: Update the cash available in each
# 1. The remaining cash from the previous year (A[t-1]), after subtrac
# 2. Interest applied to that cash, based on the interest rate for the
# 3. Bond income for the year, calculated by multiplying the purchased
m.addConstrs(
    (
        A[t] == (A[t-1] - payments[t-1]) * (1 + interest_rates[t-1]) +
        for t in range(1, T)  # Loop over each year from Year 2
    ),
    name="Yearly Cash Flow Constraints"  # Label for cash availability
)

# Constraints to ensure the cash available meets pension payment requi
# Making sure that the cash available (A[t]) is enough to cover pensio
m.addConstrs(
    (
        A[t] >= payments[t] for t in range(T)  # Ensuring cash covers
    ),
    name='Pension Payment Coverage'  # Label for constraints related t
)

# Set the objective function to minimize the initial cash (C)
# Goal: Minimize the initial cash needed while satisfying all constrai
m.setObjective(
```

```
    C,
    GRB.MINIMIZE  # Minimizing the initial cash requirement
)

# Solve the optimization
# Gurobi will find the best solution by minimizing the initial cash ne
m.optimize()

# Print the minimum initial cash required.
print(f"Optimal objective value: ${round(m.objVal, 2)}")

# Display solution values, including bond purchases and cash availabil
get_solution(m)

# Show shadow prices (dual values) for each constraint, giving insight
# if the constraints (like payments) are adjusted.
get_shadow(m)

# Show reduced costs for each decision variable, indicating potential
# if certain non-basic variables were added to the solution.
get_rc(m)
```

```
    Pension Payment Coverage[9]: 0.0, (RHS = 30000.0), (LB = -inf),
(UB = 53626.18)
    Pension Payment Coverage[10]: 0.2929907210462065, (RHS = 31000.
0), (LB = 8732.85), (UB = 116380.76)
    Pension Payment Coverage[11]: 0.0, (RHS = 31000.0), (LB = -inf),
(UB = 87543.55)
    Pension Payment Coverage[12]: 0.0, (RHS = 31000.0), (LB = -inf),
(UB = 59271.77)
    Pension Payment Coverage[13]: 0.06890446273898143, (RHS = 31000.
0), (LB = 4096.12), (UB = 61097.09)
    Pension Payment Coverage[14]: 0.33419533698546783, (RHS = 31000.
0), (LB = -0.0), (UB = 101501.46)
Reduced Cost Information
    I[0]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
    I[1]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
    I[2]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
    C: 0.0, (Coef LB = 0.0), (Coef UB = inf)
    A[0]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
    A[1]: 0.0, (Coef LB = 0.0), (Coef UB = inf)
```

Questions

Base case

Answer the following questions using only the base case model.

1. What are the obtimal Year 1 decision values, i.e. what are the values for C, and I_0, I_1, and I_2? (2.5)
    - Answer: I[0]: 73.69479810 bonds I[1]: 77.2083720 bonds I[2]: 28.83720 bonds C: $ 197,768.39680
2. In what years would it be valuable to reduce the pension payment needed, why?
    - Answer: In Years 5, 11, 14, and 15, it would be beneficial to consider reducing pension payments in the Base Case. We determine this by examining the shadow prices associated with the pension payment constraints. A shadow price shows how the objective—minimizing the initial cash required—would be impacted if the

pension payment for a particular year were adjusted by one unit. When a shadow price is positive, it means that lowering the pension payment for that year would reduce the initial cash needed, making it advantageous to decrease the payment in those years. In the Base Case, the years that hold this potential are Year 5, Year 11, Year 14, and Year 15, as these years show positive shadow prices. Reducing pension payments in these years would directly decrease the initial cash allocation required to meet pension commitments.

3. For year 15, if the payment is reduced to zero, what would be the change in the objective function value? Give your answer in a formulaic format.
    - Answer: In order to find how the objective function changes if the pension payment for Year 15 is reduced to zero, we use the shadow price for Year 15. This shadow price indicates how much the initial cash (C) changes for each dollar change in the pension payment for that year.

The formula for calculating the impact on the objective function is:

Change in Objective Function = (Reduction in Year 15 Payment)×(Shadow Price for Year 15)

With a shadow price of 0.3594 for Year 15 and a current pension payment of 31,000, reducing the payment to zero means the reduction in Year 15's payment is 31,000.

So: Change in Objective Function= 31,000×0.3594=11,141.40

Reducing Year 15's pension payment to zero would lower the initial cash requirement C C by $11,141.40.

4. For year 15, what is the maximum increase in payment value to keep the current basis solution optimal?
    - Answer: We determined the maximum increase in Year 15's payment that maintains the current solution's optimality, we look at the upper bound (UB) for the right-hand side of the Year 15 pension payment constraint. The current pension payment for Year 15 is $31,000, and the upper bound is$101,501.46. This means the payment could be increased up to $101,501.46 without changing the optimal solution. Maximum Increase = Upper $501.46 - 31,000 = 70,501.46 Therefore, the maximum allowable increase in t$ 70,501.46.

Bad case comparison

Answer the following questions using both models.

5. How much more cash is needed for the optimal solution of the bad case?
    - Answer: We can calculate the difference in cash requirement between the Bad Case and the Base Case using this formula:

Difference in Cash = Bad Case Optimal Initial Cash Value - Base Case Optimal Initial Cash Value

Base Case Objective Value: $197,768.40 Bad Case Objective Value$:232,048.36 Difference in Cash = 232,048.36 - 197,768.40 = 34,279.96

The difference in required cash between the two cases is $34,279.96.

6. What is the most invested in bond in the bad case?

- Answer: The most invested bond in the Bad Case is Bond 0 I[0], with 105.43 units purchased.

7. Do the years that are valuable for reducing pension payments change?
   - Answer: The years valuable for reducing pension payments remain the same in both the Base Case and the Bad Case. Years 5, 11, 14, and 15 continue to be

In [ ]: