Appendix A  Details for Inventory Management Example

For reference, in this section we present the full algebraic and ROME models for the inventory management example.

\[
\begin{align*}
\min_{x(\cdot), y(\cdot)} & \quad \sup_{P \in \mathcal{F}} \mathbb{E}_P \left( c' x(\tilde{z}) + h' (y(\tilde{z}))^+ \right) \\
\text{s.t.} & \quad \sup_{P \in \mathcal{F}} \mathbb{E}_P \left( y_t(\tilde{z})^- \right) \leq (1 - \beta_t) \mu_t \quad \forall t \in [T] \\
& \quad y_t(\tilde{z}) = x_t(\tilde{z}) - \tilde{z}_t \quad \forall t \in \{2, \ldots, T\} \\
& \quad 0 \leq x(\tilde{z}) \leq x^{MAX} \\
& \quad x_t \in \mathcal{L}(1, T, [t-1]) \quad \forall t \in [T] \\
& \quad y_t \in \mathcal{L}(1, T, [t]) \quad \forall t \in [T].
\end{align*}
\]

The ROME code is

```matlab
1 % inventory_fillrate_example.m
2 % Script to model robust fillrate-constrained
3 % inventory management. Solves a linearized version
4 % of the problem using LDRs
5 %
6 %
7 % Model parameters
8 T = 5; % planning horizon
9 c = 1 * ones(T, 1); % order cost rate
10 h = 2 * ones(T, 1); % holding cost rate
11 beta = 0.95 * ones(T, 1); % min. fillrate per period
12 xMax = 60 * ones(T, 1); % max. order qty. per period
13 alpha = 0.5; % temporal autocorrelation
14 %
15 % Autocorrelation matrix
```

*Stanford Graduate School of Business and NUS Business School, National University of Singapore, Email: joelgoh@stanford.edu, joelgoh@nus.edu.sg

†NUS Business School and NUS Risk Management Institute, National University of Singapore. Email: dscsimm@nus.edu.sg. The research of the author is supported by the Singapore-MIT Alliance.
L = alpha * tril(ones(T), -1) + eye(T);

% Numerical uncertainty parameters
zMax = 60*ones(T, 1);  % maximum demand per period
mu = 30*ones(T, 1);  % mean demand in each period
S = 20*(L * L');  % temporal demand covariance

% Begin model
hmodel = rome_begin('Inventory Management');

% Declare uncertainties
newvar z(T) uncertain;  % declare an uncertain demand
rome_constraint(z >= 0);
rome_constraint(z <= zMax);  % set the support
z.set_mean(mu);  % set the mean
z.Covar = S;  % set the covariance

% Define LDRs
% allocate an empty variable array
newvar x(T) empty;
% iterate over each period
for t = 1:T
    % construct the period t decision rule
    newvar xt(1, z(1:(t-1))) linearrule;
    x(t) = xt;  % assign it to the tth entry of x
end

% allocate an empty variable array
newvar y(T) empty;
% iterate over each period
for t = 1:T
    % construct the period t decision rule
    newvar yt(1, z(1:t)) linearrule;
    y(t) = yt;  % assign it to the tth entry of y
end

% Inventory balance constraints
% Period 1 inventory balance
rome_constraint(y(1) == x(1) - z(1));
% iterate over each period
for t = 2:T
    % period t inventory balance
    rome_constraint(y(t) == y(t-1) + x(t) - z(t));
end

% order quantity constraints
rome_constraint(x >= 0);  % order qty. lower limit
rome_constraint(x <= xMax);  % order qty. upper limit
Code Segment 1: ROME code for the fill rate constrained robust inventory management problem

An equivalent vectorized code in ROME that is more concise but less intuitive, is
Code Segment 2: Vectorized ROME code for the fill rate constrained robust inventory management problem

This code uses the `rome_box` contraction also used in the other examples to combine the upper and lower constraints into a single statement. It also makes comparatively heavier use of MATLAB array construction and manipulation functions such as `eye` (constructs an identity matrix), `ones` (constructs an array of all ones), `diag` (constructs a diagonal matrix), `tril` (extracts the lower triangular part of a matrix), `true` and `false` (constructs logical 0-1 matrices).
Appendix B  Details for Project Crashing Example

The full algebraic model for the project crashing problem is

\[
\begin{align*}
\min_{x(\cdot), y(\cdot)} & \quad \sup_{P \in \mathcal{F}} E_P (x_M(\tilde{z})) \\
\text{s.t.} & \quad x_j(\tilde{z}) - x_i(\tilde{z}) \geq (\tilde{z}_k - y_k(\tilde{z}))^+ \quad \forall k \in [N], A_{ik} = -1, A_{jk} = 1 \\
& \quad c' y(\tilde{z}) \leq B \\
& \quad 0 \leq y(\tilde{z}) \leq u \\
& \quad x \geq 0 \\
& \quad x_i \in L(1, N, I^i_2) \quad \forall i \in [M] \\
& \quad y_k \in L(1, N, I^k_y) \quad \forall k \in [N].
\end{align*}
\]

We consider a numerical instance of a project crashing problem with an AOA project network depicted in Figure B.1. The corresponding modeling code in ROME is

```matlab
1 % project_crashing_example
2 % Solves a project crashing example
3 % parameters
4 M = 6; % number of nodes
5 N = 9; % number of arcs
6 A = spalloc(M, N, 2*N); % allocate space for matrix
```

Figure B.1: Example AOA Project Network. Arcs are labeled with activity indices and a bracketed triplet: [optimistic activity time, mean activity time, pessimistic activity time].
% build the incidence matrix
A([1, 2], 1) = [-1; 1];
A([1, 4], 2) = [-1; 1];
A([1, 3], 3) = [-1; 1];
A([2, 5], 4) = [-1; 1];
A([2, 4], 5) = [-1; 1];
A([3, 4], 6) = [-1; 1];
A([5, 6], 7) = [-1; 1];
A([4, 6], 8) = [-1; 1];
A([3, 6], 9) = [-1; 1];

zL = [2, 3, 1, 3, 2, 5, 1, 2, 3]'; % optimistic activity times
mu = [6, 7, 3, 5, 3, 6, 3, 3, 6]'; % mean activity times
zH = [10, 11, 5, 7, 4, 7, 5, 4, 9]'; % pessimistic activity times

Sigma = diag((zH - zL).^2/12); % covariance matrix
c = ones(N, 1); % unit crashing cost
B = 10; % project budget
u = zL; % crash limit

% ROME model
hmodel = rome_begin('Project Crashing Example');

% uncertainties
newvar z(N) uncertain; % Declare uncertainties;
rome_box(z, zL, zH); % Set support;
z.set_mean(mu); % Set mean;
z.Covar = Sigma; % Set covariance;

% Decision Rules
% y: crash amounts
newvar y(N) empty; % allocate an empty variable array
% iterate over each activity
for k = 1:N
    % get indices of dependent activities
    ind = prioractivities(k, A);
    % construct the decision rule
    newvar yk(1, z(ind)) linearrule;
    % assign it to the kth entry of y
    y(k) = yk;
end

% x: node times
newvar x(M) empty; % allocate an empty variable array
% iterate over each node
for ii = 1:M
    if(ii < M)
        % find any activity that exits this node
In the construction of the decision rules, we invoke the helper function `prioractivities`, which is a recursive numerical routine, implemented in the following code:
Code Segment 4: Implementation details for prioractivities. This returns the index of all activities prior to the current ($k^{th}$) activity.
Appendix C  Details for Portfolio Optimization Example

For completeness, we repeat the algebraic formulation of the portfolio CVaR optimization problem for a fixed parameter $\tau$, which is

$$\min_{x} \text{CVaR}_\beta(-\bar{r}'x)$$

subject to:

$$\mu'x \geq \tau$$
$$e'x = 1$$
$$x \geq 0.$$

The portfolio optimization code used in this example comprises a script, and two functions, which we list here. The `optimizeportfolio` function is the key driver which contains the ROME model.

```matlab
% OPTIMIZEPORTFOLIO(N, mu, Sigma, beta, tau)
% Computes the beta-CVaR optimal portfolio
% N : Number of assets
% mu : Mean of asset returns
% Sigma : Covariance matrix of asset returns
% beta : CVaR level
% tau : Target return

function x_sol = optimizeportfolio(N, mu, Sigma, ...
    beta, tau)

% begin the ROME environment
h = rome_begin('Portfolio Optimization');

newvar r(N) uncertain; % declare r as uncertainty
r.set_mean(mu); % set mean
r.Covar = Sigma; % set covariance

% declare a nonnegative variable x
newvar x(N) nonneg;

% objective: minimize CVaR
rome_minimize(CVaR(-r' * x, beta));

% mean return must exceed tau
rome_constraint(mu' * x >= tau);

% x must sum to 1
rome_constraint(sum(x) == 1);

% solve the model
h.solve_deflected;

% check for infeasibility / unboundedness
if(isinf(h.objective))
    x_sol = []; % assign an empty matrix
else
    x_sol = h.eval(x); % get the optimal solution
```

Code Segment 5: Function to compute the optimal portfolio for fixed $\tau$

It calls a custom function, $\text{CVaR}$, which separately models the $\beta$-CVaR, for increased code modularity.

```
% CVaR(loss, beta)
% Computes the beta-CVaR
% loss : ROME variable representing portfolio loss
% beta : CVaR-level

function cvar = CVaR(loss, beta)
    newvar v; % declare an auxiliary variable v
    cvar = v + (1 / (1 - beta)) * mean(pos(loss - v));
```

Code Segment 6: Function to compute the $\beta$-CVaR

Finally, the script, `plotcvportfolio`, instantiates various parameters and iterates through a range of target returns. It concludes by plotting the coefficient of variation (CV) against $\tau$.

```
% PLOTCVPORTFOLIO
% Script which plots the coefficients of variation of
% the beta-CVaR optimized portfolio for different
% target mean returns, tau.

rand('state', 1); % fix seed of random generator
tL = 0.05; % lower limit of target mean
tH = 0.15; % upper limit of target mean
N = 20; % number of assets
mu = unifrnd(tL, tH, N, 1); % randomly generate mu
A = unifrnd(tL, tH, N, N);
Sigma = A'*A; % randomly generate Sigma
beta = 0.95; % CVaR-level

Npts = 200; % number of points in to plot
cv = zeros(Npts, 1); % allocate result array

% array of target means to test
tau_arr = linspace(0, tH, Npts);

for ii = 1:Npts
    % Find the CVaR-optimal portfolio
    x_sol = optimizeportfolio(N, mu, Sigma, ...
        beta, tau_arr(ii));
    % Store the coefficients of variation
    if(~isempty(x_sol))
        cv(ii) = Inf;
    else
        cv(ii) = sqrt(x_sol'*Sigma*x_sol) / (mu'*x_sol);
```
```
Code Segment 7: Script to compute coefficients of variation for a uniform sample of $\tau \in (\tau_L, \tau_H)$.

The output plot for this numerical example is shown in Figure C.1.

![Plot of CV vs $\tau$](image)

Figure C.1: Plot of the coefficient of variation of the CVaR-optimized portfolios against $\tau$. 

```matlab
plot(tau_arr, cv); % plot CV against tau
xlabel('\tau'); ylabel('CV'); % label axes
title('Plot of CV vs \tau'); % label graph
```