Online appendix of "Robust Optimization Made Easy with ROME"

Joel Goh *　　　Melvyn Sim †

August 2010

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.

$$
\begin{array}{lll}
\min\limits_{\boldsymbol{x}(\cdot),\boldsymbol{y}(\cdot)} & \sup\limits_{\mathbb{P}\in\mathbb{F}} \mathrm{E}_{\mathbb{P}}\left(\boldsymbol{c}'\boldsymbol{x}(\tilde{\boldsymbol{z}}) + \boldsymbol{h}'\left(\boldsymbol{y}(\tilde{\boldsymbol{z}})\right)^{+}\right) & \\
\text{s.t.} & \sup\limits_{\mathbb{P}\in\mathbb{F}} \mathrm{E}_{\mathbb{P}}\left(y_t(\tilde{\boldsymbol{z}})^{-}\right) \leq (1-\beta_t)\mu_t & \forall t \in [T] \\
& y_1(\tilde{\boldsymbol{z}}) = x_1(\tilde{\boldsymbol{z}}) - \tilde{z}_1 & \\
& y_t(\tilde{\boldsymbol{z}}) = y_{t-1}(\tilde{\boldsymbol{z}}) - x_t(\tilde{\boldsymbol{z}}) - \tilde{z}_t & \forall t \in \{2,\ldots T\} \\
& \boldsymbol{0} \leq \boldsymbol{x}\left(\tilde{\boldsymbol{z}}\right) \leq \boldsymbol{x}^{MAX} & \\
& x_t \in \mathcal{L}(1, T, [t-1]) & \forall t \in [T] \\
& y_t \in \mathcal{L}(1, T, [t]) & \forall t \in [T]\,.
\end{array}
\tag{A.1}
$$

The ROME code is

```
% inventory_fillrate_example.m
% Script to model robust fillrate-constrained
% inventory management. Solves a linearized version
% of the problem using LDRs
%

% Model parameters
T = 5;                   % planning horizon
c = 1 *ones(T, 1);       % order cost rate
h = 2*ones(T, 1);        % holding cost rate
beta  = 0.95*ones(T, 1); % min. fillrate per period
xMax  = 60*ones(T, 1);   % max. order qty. per period
alpha = 0.5;             % temporal autocorrelation

% 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
```

```

% fill rate constraint
rome_constraint(mean(neg(y)) <= mu - mu .* beta);

% model objective
rome_minimize(c'*mean(x) + h'*mean(pos(y)));

% solve and display optimal objective
% hmodel.solve;           % solve using LDR
hmodel.solve_deflected;   % solve using BDLDR

hmodel.objective       % disp. optimal objective
x_sol = hmodel.eval(x) % disp. optimal ordering rule
```

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

```
% begin model
hmodel = rome_begin('Vectorized Inventory Management');

% declare uncertainties
newvar z(T) uncertain;  % declare an uncertain demand
rome_box(z, 0, zMax)    % set distributional support
z.set_mean(mu);         % set mean
z.Covar = S;            % set covariance

% define dependency patterns for LDRs
pX = [tril(true(T)), false(T, 1)];
pY = [true(T, 1), tril(true(T))];

% define LDRs
newvar x(T, z, 'Pattern', pX) linearrule;
newvar y(T, z, 'Pattern', pY) linearrule;

% inventory balance constraints
D = eye(T) - diag(ones(T-1, 1), -1); % make a differencing matrix
rome_constraint(D*y == x - z);  % inventory balance constraint

% order quantity constraints
rome_box(x, 0, xMax);

% fill rate constraint
rome_constraint(mean(neg(y)) <= mu - mu .* beta); % fill rate constraint

% objective
rome_minimize(c'*mean(x) + h'*mean(pos(y)));  % model objective

% solve and display optimal objective
hmodel.solve;               % solve using LDR
```

```
% hmodel.solve_deflected;   % solve using BDLDR

hmodel.objective            % display optimal objective
x_sol = hmodel.eval(x)      % display optimal ordering decision rule
```

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

$$
\begin{array}{lll}
\displaystyle\min_{\boldsymbol{x}(\cdot),\boldsymbol{y}(\cdot)} & \displaystyle\sup_{\mathbb{P}\in\mathbb{F}} \mathrm{E}_{\mathbb{P}}\left(x_M(\tilde{\boldsymbol{z}})\right) & \\
\text{s.t.} & x_j(\tilde{\boldsymbol{z}}) - x_i(\tilde{\boldsymbol{z}}) \geq (\tilde{z}_k - y_k(\tilde{\boldsymbol{z}}))^+ & \forall k \in [N], A_{ik} = -1, A_{jk} = 1 \\
& \boldsymbol{c}'\boldsymbol{y}(\tilde{\boldsymbol{z}}) \leq B & \\
& \boldsymbol{0} \leq \boldsymbol{y}(\tilde{\boldsymbol{z}}) \leq \boldsymbol{u} & \\
& \boldsymbol{x} \geq \boldsymbol{0} & \\
& x_i \in \mathcal{L}(1, N, I_x^i) & \forall i \in [M] \\
& y_k \in \mathcal{L}(1, N, I_y^k) & \forall k \in [N].
\end{array}
\tag{B.1}
$$

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

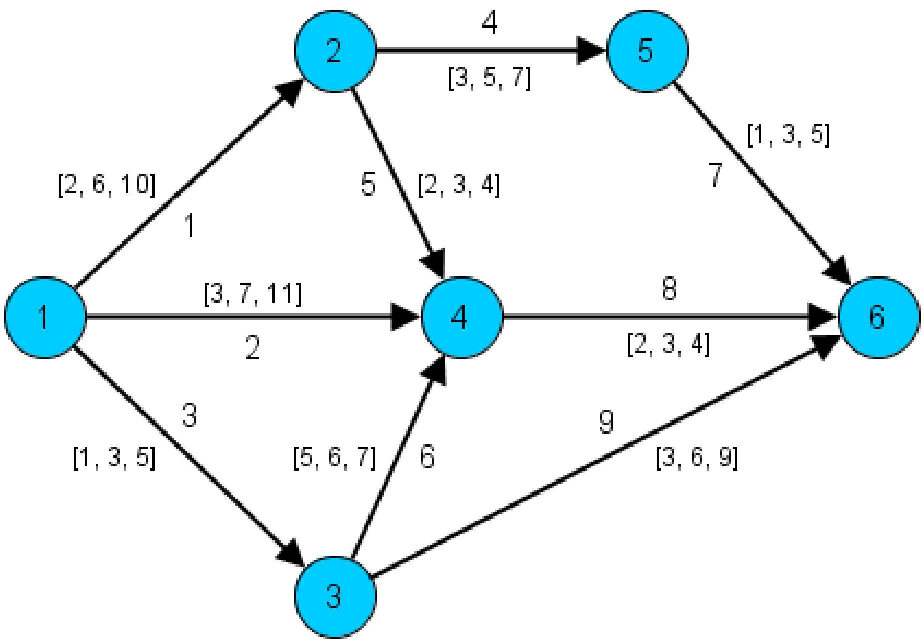

Figure B.1: Example AOA Project Network. Arcs are labeled with activity indices and a bracketed triplet: [optimistic activity time, mean activity time, pessimistic activity time].

```
% project_crashing_example
% Solves a project crashing example

% parameters
M = 6;  % number of nodes
N = 9;  % number of arcs
A = spalloc(M, N, 2*N); % allocate space for matrix

```

```matlab
% 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
```

```
    k = find(A(ii, :) < 0, 1);
    % get indices of dependent activities
    ind = prioractivities(k, A);
    % construct the decision rule
    newvar xi(1, z(ind)) linearrule;
  else
    % construct the decision rule
    newvar xi(1, z) linearrule;
  end
  x(ii) = xi;
end

% time evolution constraint
% iterate over each activity
for k = 1:N
  ii = find(A(:, k) == -1); % activity k leaves node
  jj = find(A(:, k) == 1);  % activity k enters node
  % make constraint
  rome_constraint(x(jj) - x(ii) >= pos(z(k) - y(k)));
end

% other constraints
rome_box(y, 0, u);              % crash limit
rome_constraint(x >= 0);        % nonnegative time
rome_constraint(c' * y <= B); % budget constraint

% objective: minimize worst-case mean completion time
rome_minimize(mean(x(M)));

% hmodel.solve;           % solve using LDRs
hmodel.solve_deflected;  % solve using BDLDRs
y_sol = hmodel.eval(y);  % extract crashing rule
hmodel.objective         % display optimization obj.

% simulate
rand('state', 1);       % set seed of random generator
Nsims = 10000;          % number of simulation runs
expenditure = zeros(Nsims, 1);  % allocate array
for ii = 1:Nsims
    % simulate activity times
    z_vals = zL + (zH - zL) .* rand(N, 1);
    % instantiate crash costs
    expenditure(ii) = c' * y_sol.insert(z_vals);
end
```

Code Segment 3: ROME code for the robust project crashing problem

In the construction of the decision rules, we invoke the helper function `prioractivities`, which is a recursive numerical routine, implemented in the following code

```
% PRIORACTIVITIES
% Given an AOA incidence matrix A, and an arc index k, returns the indices
% of all activities that are predecessors of k

function ind = prioractivities(k, A)
    ind = [];                              % make an empty array
    ind = rec_prioractivities(k, A, ind); % call a recursive helper


% REC_PRIORACTIVITIES
% Helper function that recursively gets the indices of the prior activites

function ind = rec_prioractivities(k, A, ind)
    % finds the unique exit node for this arc
    exitnode = find(A(:, k) < 0);

    % terminal condition
    if(exitnode == 1)
        return;
    else
        % add the immediate predecessor activities
        immediate_prior_activites = find(A(exitnode, :) > 0);
        ind = [ind, immediate_prior_activites];

        % recurse over all immediate prior activites
        for l = immediate_prior_activites
            ind = rec_prioractivities(l, A, ind);
        end
    end
```

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 τ, which is

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & \mathrm{CVaR}_\beta(-\tilde{\boldsymbol{r}}'\boldsymbol{x}) \\ \text{s.t.} & \boldsymbol{\mu}'\boldsymbol{x} \geq \tau \\ & \boldsymbol{e}'\boldsymbol{x} = 1 \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array}$$

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.

```
% 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
```

```
end
rome_end;                % end the ROME environment
```

Code Segment 5: Function to compute the optimal portfolio for fixed τ

It calls a custom function, `CVaR`, which separately models the β-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 auxilliary variable v
   cvar = v + (1 / (1 - beta)) * mean(pos(loss - v));
```

Code Segment 6: Function to compute the β-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 τ.

```
% 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 coeffients of variation
  if(isempty(x_sol))
    cv(ii) = Inf;
  else
    cv(ii) = sqrt(x_sol'*Sigma*x_sol) / (mu'*x_sol);
```

```
  end
end

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

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.

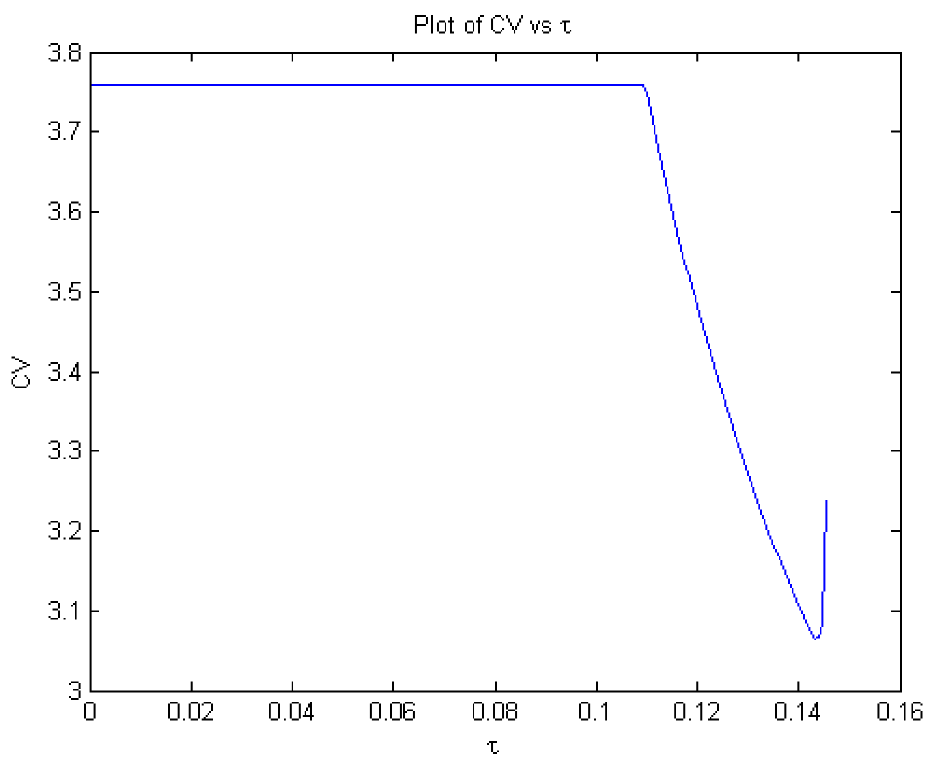

Figure C.1: Plot of the coefficient of variation of the CVaR-optimized portfolios against τ.