Markowitz Portfolio Example

Mac Radigan

0.1 Introduction

Markowitz portfolio theory is used in investment management as a tool for diversifying away risk through portfolio balancing. Given a set of investment assets, it finds the most efficient allocation (portfolio weights). Here, efficiency is defined as minimizing the expected variance. Markowitz portfolios may be subject to specified constraints, such as a specific return on investment (expected price), non-negativity constraints (restricting short selling), and may include risk-free assets or market indexes [1].

This example will select a minimum variance portfolio, constrained to a specified expected rate of return. All results will be annualized as commonly reported to investors.

0.2 Implementation

This optmization problem may be expressed as minimizing the portfolio variance, subject to the constraint that the individual allocation weights of the stocks add to unity, and the expected return is equal to the specified target.

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \sigma_{p,w}^2 = \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} \\ \text{subject to} \quad & \mu_{opt} = \mathbf{w}^T \mu \\ & \mathbf{w}^T \underline{\mathbf{1}} = 1 \end{aligned}$$

Where

$$
\begin{array}{rl}
\sigma^2_{p,w} & \text{portfolio variance of weighted assets} \\
\mathbf{w} & \text{individual asset weights} \\
\mathbf{\Sigma} & \text{covariance matrixw} \\
\mu & \text{individual asset returns} \\
\mu_{opt} & \text{target portfolio return}
\end{array} \tag{1a}
$$

Forming the Lagrangian function for the constrained minimization, we have

$$L(w, \lambda_1, \lambda_2) = \mathbf{w}^T \mathbf{\Sigma} \mathbf{w} + \lambda_1 (\mathbf{w}^T \mu - \mu_{opt}) \tag{2}$$

So the first order conditions are

$$2\partial L(w, \lambda_1, \lambda_2) = 2\mathbf{\Sigma}\mathbf{w} + \lambda_1 \mu + \lambda_2 \underline{\mathbf{1}} = \underline{\mathbf{0}} \tag{3}$$

$$\frac{2\partial L(w, \lambda_1, \lambda_2)}{\partial \lambda_1} = \mathbf{w}^T \mu - \mu_{opt} = 0 \tag{4}$$

$$\frac{2\partial L(w, \lambda_1, \lambda_2)}{\partial \lambda_2} = \mathbf{w}^T \underline{\mathbf{1}} - 1 = 0 \tag{5}$$

Expressed in matrix form

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 2\mathbf{\Sigma} & \mu & \underline{\mathbf{1}} \\ \mu^T & 0 & 0 \\ \underline{\mathbf{1}}^T & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{w} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{0}}^T \\ \mu_{opt} \\ 1 \end{bmatrix} = \mathbf{b} \tag{6}$$

This may be solved for $\mathbf{b}$, where the first $n-2$ elements of $\mathbf{b}$ are the portfolio weights ($\mathbf{w}_n = \mathbf{b}_n$).

$$\mathbf{b} = \mathbf{A}^{-1}\mathbf{x} \tag{7}$$

References

[1] Wikipedia, "Modern portfolio theory — Wikipedia, the free encyclopedia," 2011, [Online; accessed 13-December-2013]. [Online]. Available: http://en.wikipedia.org/Modern_portfolio_theory

1 Appendix A: The Example Portfolio

1.1 Portfolio #1: AAPL, JPM, LMT, XOM

Portfolio #1 (AAPL, JPM, LMT, XOM)			
Symbol	**Weight**	**ROI**	**Volatilty**
AAPL	0.08	11.4%	28.8%
JPM	0.12	31.6%	19.1%
LMT	0.39	55.8%	16.3%
XOM	0.40	08.2%	13.0%
portfolio		25.0%	11.76%

Table 1: Portfolio #1

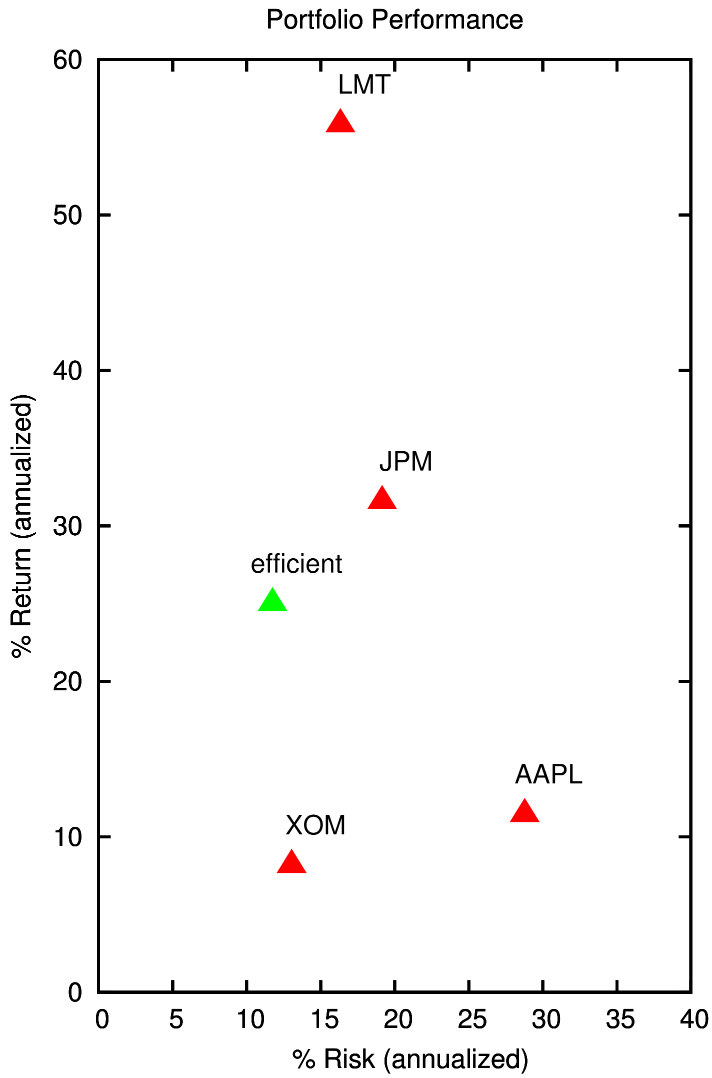

Figure 1: Portfolio Performance

2 Appendix B: Historical Data

2.1 AAPL Stock Chart

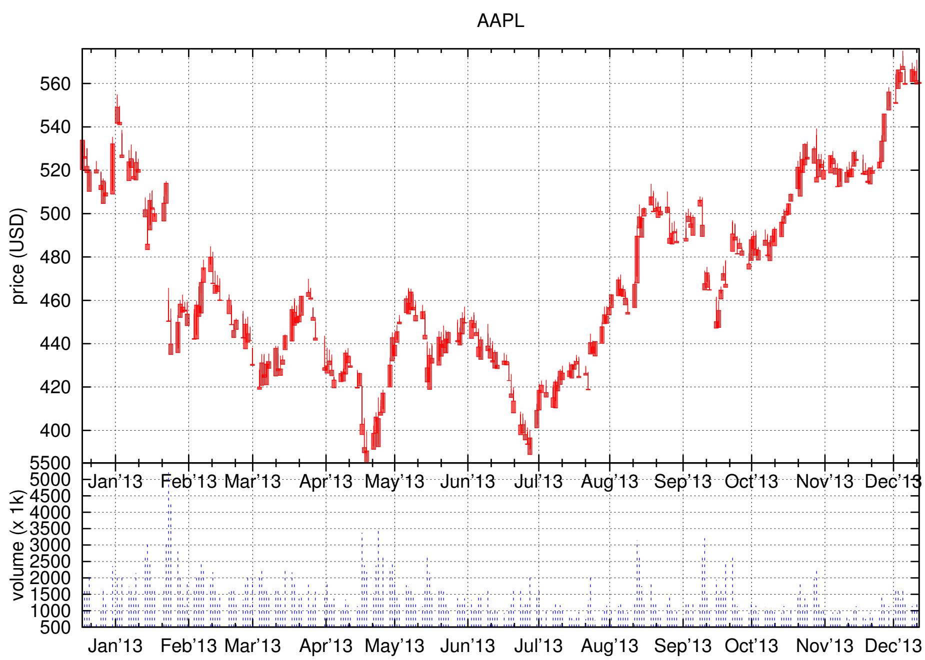

Figure 2: AAPL

2.2 JPM Stock Chart

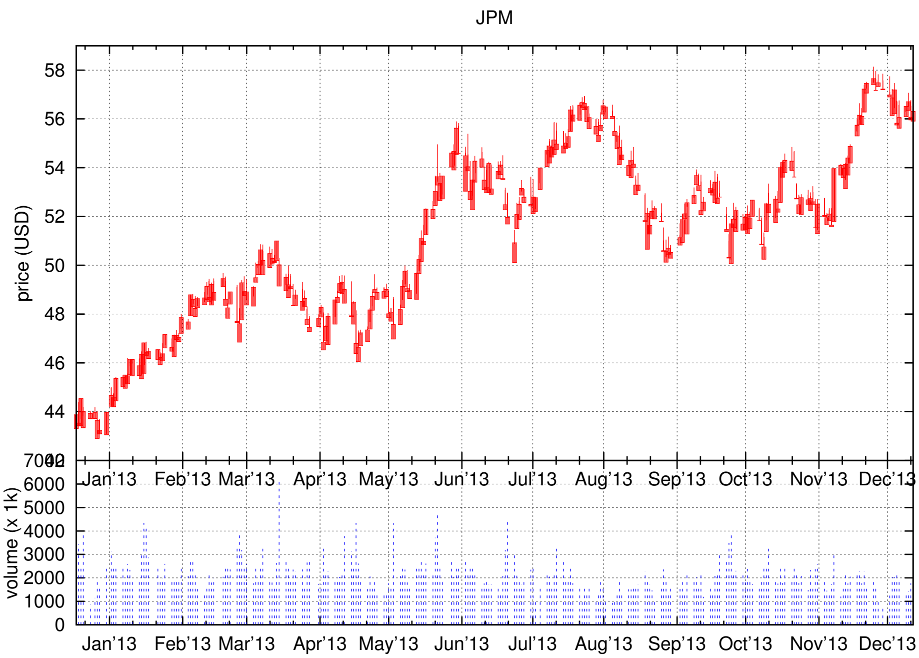

Figure 3: JPM

2.3 LMT Stock Chart

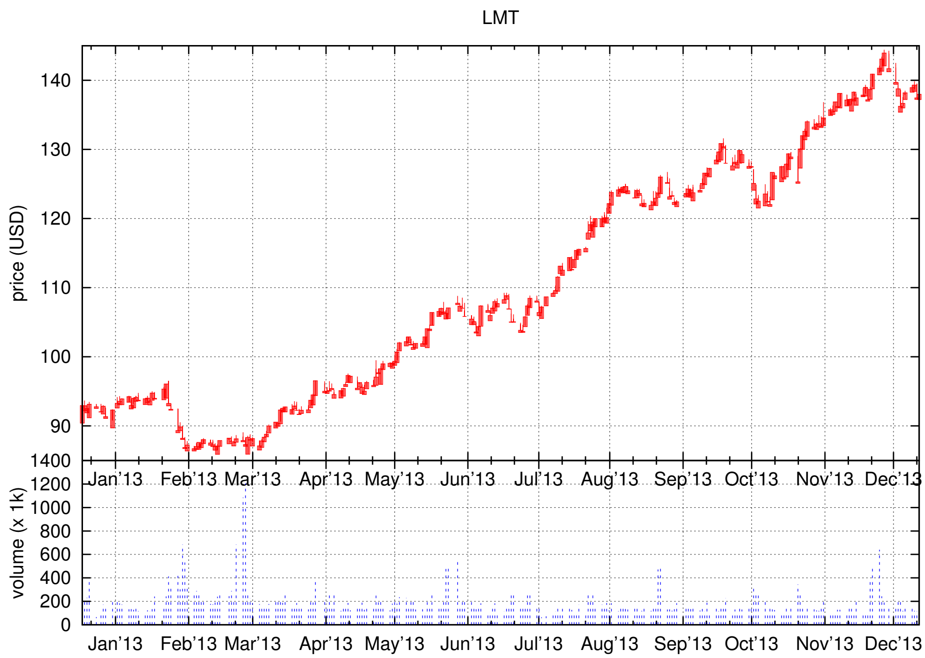

Figure 4: LMT

2.4 XOM Stock Chart

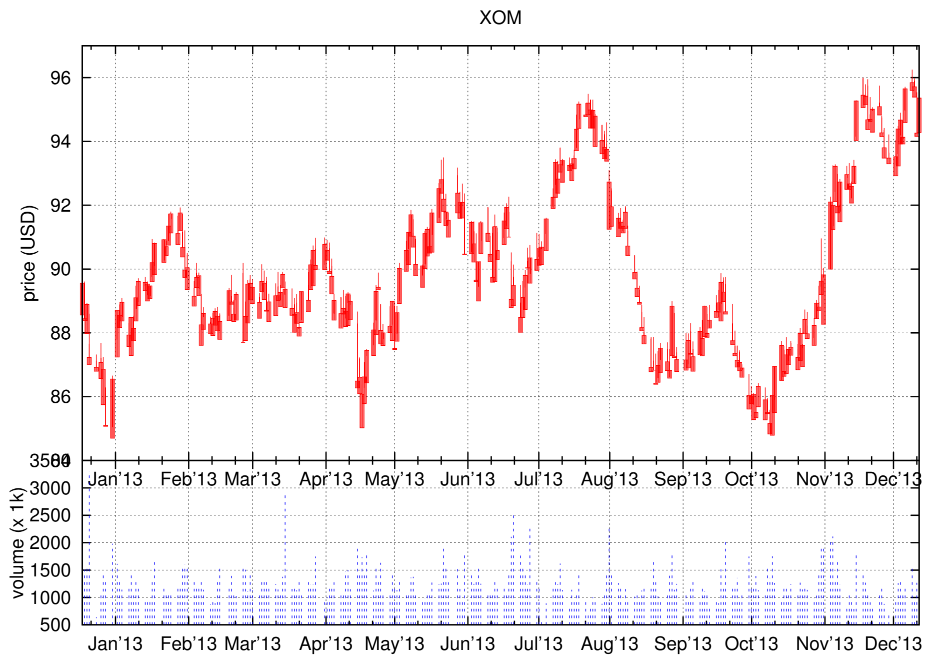

Figure 5: XOM

3 Appendix C: Octave Source Code

3.1 Appendix C.1: script example (main entry point)

```
#!/usr/bin/env octave
%#!/usr/bin/env octave -qf
% MARKOWITZDEMO  runs the Markowitz demo
%
%  Description:  This is a command line entry point for
%                demonstrating portfolio balancing using
%                a basic applicaton Markowitz portfolio theory.
%
%                In the current implementation, a target return
%                on investments is specified, short selling is
%                permitted, and there is no inclusion of a risk-free
%                asset.
%
%  See Also:     http://wikipedia.org/wiki/Modern_portfolio_theory
%                for more information on the Markowitz portfolio
%
%                http://finance.yahoo.com
%                for more information on Yahoo! Finance data sources
%
%  Author:
%    Mac Radigan

return_opt    = 25.0;     % desired annualized rate of return (percent)
window_length = 250;      % window size (in days) of stock price samples
                          % 250 = 1 year of trading days
symbols = sort({    ...   % stocks in portfolio
            'AAPL', ...
            'JNJ',  ...
            'LMT',  ...
            'XOM',  ...
           })';
SAVE_FIGURES = true;      % enable to save figures {T:enable, F:disable}

[weights, volatility, volatility_opt, portfolio_rreturn, rate_of_return] ...
  = markowitzPortfolio(symbols, window_length, return_opt);
```

```

% print the efficient portfolio results as a table
fprintf(stdout, 'efficient portfolio: return=%2.2f%%, ...
    volatility=%2.2f%%\n',portfolio_rreturn,volatility_opt);
% load all tabular data into a cell array
D=vertcat( ...
  symbols', num2cell(weights)', ...
  num2cell(rate_of_return), num2cell(volatility));
fprintf(stdout, '\tSYM\tWEIGHT\tRETURN\tVOLATILITY\n',D{:});  % print tabular data
fprintf(stdout, '\t%s\t%2.2f\t%2.2f%%\t%2.2f%%\n',D{:});  % print tabular data

% plot results
if(SAVE_FIGURES)
  FS=15;
  FW='bold';
  mxy = [max([volatility volatility_opt]), max([rate_of_return return_opt])];
  dxy = 1.5*[1 1];
  %hfig = figure('visible','off');
  hfig = figure('visible','on');
  scatter(volatility,rate_of_return,40,'red','^');
  hold on;
  text(volatility+dxy(1),rate_of_return+dxy(2),symbols);
  scatter(volatility_opt,return_opt,40,'green','^');
  text(volatility_opt+dxy(1),return_opt+dxy(2),'efficient');
  title('Portfolio Performance','FontSize',FS,'fontweight',FW);
  ylabel('Annualized Return (\mu)','FontSize',FS,'fontweight',FW);
  xlabel('Annualized Risk (\sigma)','FontSize',FS,'fontweight',FW);
  legend('Efficient Portfolio');
  axis([0 mxy(1)],[0 mxy(2)]);
  outdir='results';
  fmts={'eps','jpg','png','tiff'};
  name=sprintf('portfolio_performance_%s',strjoin('_',symbols));
  mkdir(outdir);
  for fmt=fmts
    filename=sprintf('%s/%s.%s',outdir,name,fmt{:});
    fprintf(stdout,'\tsave> %s\n',filename);
    saveas(hfig,filename);
  end
end

%% *EOF*
```

3.2 Appendix C.2: Markowitz Portfolio optimization function

```
% MARKOWITZPORTFOLIO minimum-variance Markowitz portfolio solver
%
%  [weights, volatility, volatility_opt, portfolio_rreturn, rate_of_return] ...
%    = markowitzPortfolio(symbols, window_length, return_opt);
%
%    where
%            symbol          is a cell array of ticker symbols,
%                            i.e. {symbol1, symbol2, ...}
%
%            window_length   is the number of samples used in
%                            risk/return estimation
%
%            return_opt      is the desired annual portfolio percent return
%
%            weights         is a Nx1 real matrix of the Markowitz portfolio
%                            allocation weights, one for each ticker symbol
%
%            weights         is a Nx1 real matrix of the Markowitz portfolio
%                            allocation weights, one for each ticker symbol
%
%            volatilities    is a Nx1 real matrix of the volatility of
%                            asset in the portfolio, one for each ticker symbol
%
%            volatility_opt  is the volatility of the Markowitz-weighted
%                            portfolio
%
%            rate_of_return  is a Nx1 real matrix of annualized rates of return
%                            of each asset in the portfolio, one for each
%                            ticker symbol
%
%  Description:  This class identifies the minimum-variance
%                Markowitz portfolio, for a specified set of
%                portfolio stocks and target portfolio return
%                on investment.
%
%  Examples:
%
```

```
%    return_opt    = 25.0;     % desired annualized rate of return (percent)
%    window_length = 250;      % window size (in days) of stock price samples
%                              % 250 = 1 year of trading days
%    symbols = sort({    ...   % stocks in portfolio
%                'AAPL', ...
%                'JNJ',  ...
%                'LMT',  ...
%                'XOM',  ...
%               })';
%
%    [weights, volatility, volatility_opt, rate_of_return] ...
%        = markowitzPortfolio(symbols, window_length, return_opt);
%
%
%  See Also:     http://wikipedia.org/wiki/Modern_portfolio_theory
%                for more information on the Markowitz portfolio
%
%  Author:
%    Mac Radigan

function [weights, volatility, volatility_opt, portfolio_rreturn, rate_of_return, S, R] = ...
    markowitzPortfolio(symbols, window_length, return_opt=NaN)
  volatility = [];        % volatility
  rate_of_return = [];    % rates of return
  weights = [];           % portfolio weights
  volatility_opt = NaN;   % portfolio volatility
  %
  T = 250;   % number of trading days in a year (time horizon)
             % (either 250 or 252, depending specific reporting requirements)
  P = 1/T;   % time period
  Np = numel(symbols);         % number of stocks in portfolio
  Ns = window_length-1;        % number of samples
  % X is an M x N matrix of daily returns,
  %    where M is the number daily returns
  %    and   N is the number number of stocks in the portfolio
  X = zeros(Ns,Np);
  for idx=1:numel(symbols)
    % retreive stock data
    ds = getStockData(symbols{idx});
    ds2.(symbols{idx}) = ds;
    x_close   = ds.Close(1:window_length);        % limit samples to window_length
```

```
    X(:,idx)  = -1*diff(x_close)./x_close(2:end); % rates of return (daily)
  end
  R = corrcoef(X);      % correlation matrix of daily returns
  S = cov(X,1);         % covariance matrix of daily returns
  mu = mean(X);         % average daily rate of return
  sig = sqrt(diag(S))'; % variance of daily return
  %
  % If a target rate of return is specified, perform a perfolio optimization with
  % this constraint.  Otherwise, use the global minimum variance solution.
  %
  if(isnan(return_opt)) % global minimum-variance Markowitz portfolio
    [w, mu_opt, sig_opt] = MPglobalMinimumVariance(mu, Np, S);
  else                  % Markowitz portfolio with specified rate of return
    [w, mu_opt, sig_opt] = MPconstrainedReturn(mu, S, Np, T, return_opt);
  end
  % return values: portfolio weights, portfolio volatility, and
  %                individual rates of return and volatilities
  % retport all return/volatility as annualized percentages
  weights = w;                                 % rename rates
  volatility = sig/sqrt(1/T)*100;              % annualized volatility (%)
  volatility_opt = sig_opt/sqrt(1/T)*100;      % annualized portfolio variance (%)
  rate_of_return = ((1+mu).^T-1)*100;          % annualized portfolio
                                               % rate of return (%)
  portfolio_rreturn = ((1+mu_opt).^T-1)*100; % annualized target return (%)
end

function [w, mu_opt, sig_opt] = MPglobalMinimumVariance(mu, Np, S)
  % Markowitz global minimum-variance portfolio optimization
  %
  %      minimize:    sigma_p,w^2 = w'*S*w
  %
  %      subject to:        w'*1v = 1
  %
  %      where
  %         sigma_p,w^2    portfolio variances
  %         w              portfolio weights
  %         S              covariance matrix
  %         1v             one vector
  %
  %
  %         A     *      x      =    b
```

```
  %     | 2S 1v | * | w       | = | 0v  |
  %     | 1' 0  |   | lambda2 |   | 1   |
  %
  %       where
  %         S       daily rate of return covariance matrix
  %         S       daily rate of return covariance matrix
  %         mu      daily mean ROI
  %         w       portfolio weights
  %         1v      one vector (size of portfolio)
  %         0v      zero vector (size of portfolio)
  %         lambda2 Lagrange multiplier for normalized weight constraint
  %
  % Markowitz linear equations
  A = [ 2*S ones(1,Np)' ; ones(1,Np) 0 ];
  b = [ zeros(1,Np) 1 ]';
  z = A\b;                     % solve Ax=b
  w = z(1:Np);                 % assign portfolio weights
  mu_opt   = mu*w;             % mean portfolio return
  sig2_opt = w'*S*w;           % portfolio variance
  sig_opt  = sqrt(sig2_opt); % portfolio standard deviation
end

function [w, mu_opt, sig_opt] = MPconstrainedReturn(mu, S, Np, T, return_opt)
  % Markowitz portfolio optimization with constraint on ROI
  %
  %      minimize:  sigma_p,w^2 = w'*S*w
  %
  %      subject to:     mu_opt = w'*mu
  %
  %                      w'*1v  = 1
  %
  %      where
  %         sigma_p,w^2      portfolio variances
  %         mu_opt  target   portfolio rate of return
  %         w                portfolio weights
  %         S                covariance matrix
  %         1v               one vector
  %
  %
  %           A       *     x       =     b
  %     | 2S  mu 1v | * | w       | = | 0v     |
```

```
  %     | mu' 0  0  |    | lambda1 |    | mu_opt |
  %     | 1'  0  0  |    | lambda2 |    | 1      |
  %
  %       where
  %         S       daily rate of return covariance matrix
  %         mu      daily mean ROIs
  %         mu_opt  target portfolio rate of return
  %         w       portfolio weights
  %         1v      one vector (size of portfolio)
  %         0v      zero vector (size of portfolio)
  %         lambda1 Lagrange multiplier for target return
  %         lambda2 Lagrange multiplier for normalized weight constraint
  %
  mu_opt = (1+return_opt/100)^(1/T)-1; % convert target return to fractional daily
  % Markowitz linear equations
  A = [ 2*S mu' ones(1,Np)' ; mu 0 0 ; ones(1,Np) 0 0 ];
  b = [ zeros(1,Np) mu_opt 1 ]';
  z = A\b;                          % solve Ax=b
  w = z(1:Np);                      % assign portfolio weights
  mu_opt_assert  = mu*w;            % mean portfolio return (sanity check)
  sig2_opt       = w'*S*w;          % portfolio variance
  sig_opt        = sqrt(sig2_opt); % portfolio standard deviation
  % check result
  tol = 10*eps;                            % numerical tolerance
  assert(abs(mu_opt_assert-mu_opt)<tol);  % assert
  A
  b
  z
end

%% *EOF*
```

3.3 Appendix C.3: Stock Data Reader

```
function [data] = getStockData(symbol)
% GETSTOCKDATA read Yahoo! Finance time series CSV files
%
%  getStockData(symbol)
%
```

```
%    where
%            symbol is a ticker symbol (character array)
%
%            data   is a time—series structure with the
%                   following fields:
%                     .symbol    — ticker symbol          (character array)
%                     .Date      — epoch time in seconds  (Nx1 int32 matrix)
%                     .DateStr   — localized date string  (character array)
%                     .Open      — opening price          (Nx1 real matrix)
%                     .High      — high price             (Nx1 real matrix)
%                     .Low       — low price              (Nx1 real matrix)
%                     .Close     — closing price          (Nx1 real matrix)
%                     .Volume    — volume                 (Nx1 real matrix)
%                     .Adj_Close — adjusted closing price (Nx1 real matrix)
%
%  Examples:
%    data = getStockData('LMT')
%
%  Author:
%    Mac Radigan
%
  data = {};                                            % stock data
  data.symbol = symbol;                                 % stock ticker symbol
  filename = sprintf('../data/%s/%s.csv',symbol,symbol); % path to stock data
  if(¬exist(filename,'file'))
    error('file %s not found',filename);                % assert
  end
  fid = fopen(filename,'r');                            % open for reading
  fields = regexprep(strsplit(fgetl(fid),','),' ','_');  % read field names
  fmt = '%10c,%f,%f,%f,%f,%d,%f\n';                     % CSV field pattern
  n = size(strsplit(fmt,','),2);                        % number of fields
  D = cell(n,1);                                        % temporary storage
  for fIdx=1:n, data.(fields{fIdx}) = []; end           % initialize fields
  while(!feof(fid))
   [D{:},¬] = fscanf(fid,fmt,'C');                        % read each line
   data.(fields{1}){end+1} = ...
     mktime(strptime(D{1},'%Y—%m—%d'));                   % record datetime
   for fIdx=2:n, data.(fields{fIdx})(end+1) = D{fIdx}; end % assign fields
  endwhile
  % all matrix operations are defined in terms of column vectors,
  % so transpose
```

```
  for fIdx=2:n, data.(fields{fIdx}) = data.(fields{fIdx})'; end
  % convert datetime stamp to ASCII string, using system localization
  for rIdx=2:numel(data.Date)
    data.DateStr{rIdx} = asctime(localtime(data.Date{rIdx})); % localized date
  end
  fclose(fid);
end

%% *EOF*
```