*Critical-Point*Ó for *Windows* Manual

V. 3.00 01-12-1998

Introduction

Critical-Point for Windows is an integrated software to design, solve and analyze mean-variance (Markowitz) portfolio optimization models. This software is distributed in two versions: student and professional. Both versions are almost identical. The professional version can solve large models used in profit oriented and commercial organizations. The student version is limited to solve models with at most 12 assets, 4 options and 20 constraints. The student version should be used exclusively in academic research and non profit activities. For those purposes it is freely available thanks to grants from BM&F, São Paulo Futures and Mercantile Exchange, and NOPEF-USP, Center for Optimization and Stocastic Processes Applied to Finance and Economics of the University of São Paulo. The copyrights of both versions belong to its author, who can be reached at the telephone and fax number +55-11-259-1217, at e-mail address jstern@ime.usp.br.

Installation

The minimum configuration required by the student version is an IBM compatible PC-486-DX with 4MB RAM, and 3MB free in the hard disk, MS-Windows 95. The professional version requires a Pentium-PC, 16MB RAM and 4MB free in the hard disk.

To proceed with the installation, please follow the steps:

1. Put the installation disk in Drive A (or B);

2. Execute (from Windows) A:\SETUP
where the "A" indicates the drive where the installation program is.

The student version of Critical-Point will be installed in the directory C:\SCRIPO, and the professional version in directory C:\CRIPO. You are not supposed to change the name and location of these directories in the installation process.

Note: If the installation aborts with an error message about file A.DLL or A.OCX, you ought to exit Windows, delete these files from directory C:WINDOWS\SYSTEM and restart the installation.

Starting

To start *Critical-Point* do the following:

1. Start Windows and double click the *Critical-Point* Icon;

2. Type the Password "student". Use only small caps without the quotes;

3. The *Critical-Point* menu bar will appear at the top left of the screen.

Efficient frontier generation

In order to trace Markowitz Efficient Frontier, click in sequence the buttons:

1. Filter (F),

2. include Derivatives (D),

3. MDL Model compiler (M),

4. Parametric Quadratic Programming (Q),

in the menu bar. Alternatively, you can run these programs from the Run pulldown menu. This sequence solves the model specified in the files MODEL.CP, DERIV.CP and FILTER.CP as explained later.

Efficient frontier visualization

Click the Graph button (G). The graphic of the efficient frontier will show up. The critical points used to compute the efficient frontier are shown by the symbol "+". Each critical point is characterized by some qualitative change in the portfolio, like:

- an asset enters or leaves the portfolio, or

- a constraint goes from tight (active) to slack (inactive), or vice-versa.

The titles, colors, numeric formats, visualization of tangent lines and many other options can be changed in the menu Options Graph. The user's new options will be saved automatically. Use the command Options Reset Graph to return to the default values.

Numeric template

This window offers numeric information about Efficient Frontier. To open it, click the numeric template button(N). This template has six rows and eight columns, plus the buttons Recalculate, Plot, Report and Save. The rows are named Selection A to F and can be used to analyze up to six efficient portfolios simultaneously.

The 1st column shows h, a ponderation parameter between expected return and risk of the portfolio. This is a risk propension parameter (opposite of risk aversion). The higher h, the more important the portfolio's expected return as compared to the portfolio's risk.

The 2nd column shows the portfolio's expected return (e).

The 3rd column shows the portfolio's variance (v).

The 4th column shows the portfolio's standard deviation (s).

The 5th column shows implicit interest rate (r). This is the interest rate for which a given portfolio is tangent. Alternatively, given an interest rate Critical-Point computes the corresponding tangent portfolio.

The 6th and 7th columns show parameters k and l. Parameter k tells that a portfolio is between the critical points k and k+1. Parameter l is the relative position between these two extremes, so l=0 means the portfolio is at critical point k, l=0.5 means the portfolio is half way between k and k+1, and l=1 means the portfolio is at critical point k+1.

The 8th column gives the status of the selection row. The symbol:

Æ shows that the selection was not yet recomputed;
+ means that the portfolio is at a critical point;
Ö means that the portfolio is not at a critical point;
�� indicates that the input parameter was too high or too low.

We can select a portfolio inputting the value of any of these parameters: click the mouse at the input field and enter the parameter's value (this field will appear underlined and in blue). Next click the recalculate button and all the other parameters will be recomputed consistently with the input parameter (these parameters will appear in black). Whenever a selection is recalculated, an auxilliary window will shows the composition of the selected portfolio.

The parameters e and s can also be selected directly in the graph window: use the mouse cursor to point at a selected position in the efficient frontier; then click the left button to copy the expected return parameter in the appropriate field of the selected row (use the right button to copy the s coordinate).

The Plot button draws the efficient frontier and shows the selected points with small circles on the frontier. When a portfolio is selected by its interest rate (r), also the tangent line is drawn.

The report button generates a description of all the selected portfolios written in the ASCII text file REPORT.CP. This file can be read and printed through any text file editor, which can be called by means of the Open File button in the command bar. The default editor to be called can be changed in the options menu. The save button preserves all the data before leaving the program.

The numeric format of the Numeric Template can be changed in the menu Options Numeric. To return to the default format (floating point notation), use the Option Reset Numeric command.

Printing

Click the last button in the command bar, or use the File Print Graph menu, to print the efficient frontier. The printer and the printer options to be used are the default in your windows environment. These settings can be changed with the File Print Setup menu. Using the command Options to Clipboard, the graph can be exported to the Clipboard as a Bitmap or Metafile and edited by other graphic programs.

MODEL DESCRIPTION

The DERIV.CP and MODEL.CP files contain the information about the model that must be provided by the user to *Critical-Point*. These are ASCII text files that can be edited by any text (non formating) editor. The file MODEL.CP has the names of all the assets allowed to enter a portfolio, as well as all the constraints that a portfolio must obey.

An example:

```
all={ TEL4, ELE6, PET4, BB4, BBD4, SCO4, CEV4, BRH4 };

#sectors

energy = { ELE6, PET4 };
banks = { BB4, BBD4 };
food = { SCO4, CEV4, BRH4 };
state = {TEL4, ELE6, PET4, BB4 };
private = ~state;

#constraints

normal: sum[all] $ == 1;
statelim: sum[state] $ <= 0.5;
statebanks = state & banks;
statebklim: sum[statebanks] $ <= 0.1;
foodsbound: sum[food] $ >= 0.2;

liquindex[all]={
1.0@TEL4, 0.6@ELE6, 0.5@PET4, 0.4@BB4,
0.4@BBD4, 0.2@SCO4, 0.2@CEV4, 0.3@BRH4 };

liqconstr: for[all] $ <= 0.5*liquindex;
```

The first line defines the universe set, that is all the assets that are allowed in a portfolio.

The fourth line defines the sub-set (or sector) energy, the following lines in the section define other sectors (banks, food, state, private).

The first constraint (normal) establishes that the sum of all investments in a portfolio must be equal to 1. This is the normal constraint that must always be present, indicating the total investment in the portfolio. It can be a number of monetary units, but is usually taken as 1 or 100, so that investment in each asset will appear as unit fractions or percentages of the total investment.The symbol "==" is the equality symbol, not to be confused with "=", the attribution symbol.

The second constraint establishes that the summation of investmets in *state* assets (defined as {TEL4, ELE6, PET4, BB4 }) must be less than or equal to 0.5

The line that follows the second constraint defines the set *statebanks*, containing the assets which belong to *banks* and *state* sets.

The third constraint establishes that the sum of investings in *statebanks* must be less than or equal to 0.1.

The *foodsbound* constraint establishes that sum of investings in *food* set must be equal to or greater than 0.2

liquindex is an array defined on the domain set *all*, worths 1.0 at TEL4, 0.6 at ELE6, etc. The last constraint establishes that the investing for each asset must be smaller than or equal to 50% of the value associated with the asset in *liquindex* array.

Expected returns and risks

The FILTER.CP file contains information about the expected return of each fundamental (non derivative) asset, for the period of time considered in the model, as well as the standard deviation (a measure of risk) associated to this prognostic.

```
fundamental={
TEL4, ELE6, PET4, BB4, BBD4, SCO4, CEV4, BRH4 };

std[fundamental]={
2.105123e-001@TEL4, 3.214724e-001@ELE6, 2.988641e-001@PET4,
2.952717e-001@BB4, 2.019181e-001@BBD4, 1.709987e-001@SCO4,
2.471665e-001@CEV4, 1.866201e-001@BRH4 };

er[fundamental]={
4.521883e-003@TEL4, 9.349340e-002@ELE6, 1.414101e-001@PET4,
4.441184e-002@BB4, 4.125617e-002@BBD4, 2.153917e-002@SCO4,
-1.467467e-001@CEV4, 7.254108e-002@BRH4 };

Hurst=0.50;
```

extrap=30;

sample=90;

The array *er* has the expected returns and the array *std* the standard deviations of the returns. We say that we expect from asset asseta a return of *er*[asseta] "more or less" *std*[asseta].

The DERIV.CP file may contain information to overwrite expected returns and the standard deviations (a measure of risk) of fundamental assets. DERIV.CP also contains the definition of the derivative assets to enter the model, as explained later.

When writing floating point numbers, do not forget the ZERO BEFORE THE DECIMAL POINT.

The model description language, MDL

Let us now examine in greater detail the MDL language or Model Description Language (pronounced "MoDeL"): The MDL language purpose is to describe, easy and intuitively, the constraints that the portfolio should obey.

NAMES

A valid name is a sequence of letters, digits, underlines and dots, beginning by a letter and having at most 15 characters. It is forbidden to use the same name for two or more objects in an MDL program. It is also forbiden to use aas a name a reserved word. The reserved words of the MDL language are: abs, exp, for, ln, max, min, maxe, mine, print, sign, short, sum.

INDICES

In the Markowitz type models, portfolios are optmized choosing assets in a given universe. This set of all assets under consideration --also known as the universe set-- is the set all, or the index set of the model. The definition of the universe set must be the first line in a MDL language program.

SETS

A set is defined by an attribution statement made of:

1. The name of the set.
2. The attribution symbol "=".
3. A list of asset names.
4. The termination symbol ";".

A list of asset names is made of:

1. The begin list symbol "{",
2. A sequence of asset names, separates by commas,
3. The end list symbol "}".

Besides defining sets, or sectors, by the list of its elements, sets can be defined by operations between other sets. These operations are:

~: The complement of a given set, as the set of all elements in the universe not present in the given set.
|: The union of two sets, as the set of all elements in the first or in the second or in both sets.
&: The intersection of two sets, as the set of elements in both sets.
\: The difference between two sets, as the set of elements that belong to the first but not to the second set.

In the following examples, we define some new sets through operations between preexisting sets:

private = ~state;
statebanks = state & banks;
foodandenergy=food | energy;

The precedence rules for the set operations resembles those of ordinary arithmetic: In a givem expression, first we perform the complement operations, then the intersections and finally the unions and differences. The standard evaluation order can be changed using parenthesis as punctuation symbols.

SCALARS

A scalar is a number: integer or real.

Integer numbers are sequence of one or more decimal digits: 1, 2, ... 9, 0. An integer number may be preceded by the plus or minus signs, without space between the sign and the sequence of digits. For example, these are integer numbers:

0; -998; 754

A real number can be represented in two formats:

- Floating point notation, or
- Fixed point notation.

A real number in fixed point notation is a sequence of:

- An integer, the decimal point, and an unsigned integer.

Observe that, in fixed point notation, we must have at least one digit, possibly zero, both before and after the decimal point. The following are correct examples of real numbers in fixed point notation: 0.0; 37.65; -8.83. The following are **incorrect** examples: .15; 16.; -.88

A real number in floating point notation is a sequence of:

1. An integer number or a real in fixed point notation. This is the floating point number mantissa.
2. The letter E in small or big caps.
3. An integer. This is the floating point number exponent.

The floating point number exponent indicates the number of digits, to the right or to the left, by which the decimal point in the mantissa should be moved. This notation is useful to write very large or very small numbers, without the burden of long sequences of zeros before or after the decimal point. Thus, we can write 1000000 as 1E6, -1000000 as -1E6, 0.0000001 as 1E-6 and -0.0000001 as -1E-6.

ASSOCIATIVE ARRAYS

Arrays associate numeric values to assets. Attributions to arrays are always restricted to the attribution's domain set, which is indicated at the attribution's left hand side. For example: using the sets already defined in the section as domains:

std[all]={
2.105123e-001@TEL4, 3.214724e-001@ELE6, 2.988641e-001@PET4,
2.952717e-001@BB4, 2.019181e-001@BBD4, 1.709987e-001@SCO4,
2.471665e-001@CEV4, 1.866201e-001@BRH4 };

mincapt[private] = { 0.05@CEV4, 0.03@SCO4, 0.02@BBD4 };

maxmin[private] = max(minpart , mincapt);

The expression <value>@<asset> associates the array value to the asset.

The following domain rules apply to attributions:

- An attribution does not change the values of an array outside the attribution's domain.
- When the domain is ommitted, it is taken as the universe.
- The initial value of an array in all assets is zero.

We can attribute values to arrays by means of arithmetic operations (addition, subtraction, product, division and power) between two arrays. The arithmetic operations are pointwise, i.e. they are computed element by element. Thus, for example, after the attribution

*example[private]=std*er;*

the value of the elements of *example* in *private*, is *std*er*, the value of *example* at the other elements stay unchanged, if it is the first attribution of *example*, the value at the other elements is zero.

We can also make arithmetic operations between a scalar and an array. In this case, the scalar is interpreted as a constant array.

Example:

*er3[a]=3*er;* # each element of er which belongs to a is

# multiplied by 3 and stored in er3

Finally, we can access and change an array value at a single asset. For example:

std[CEV4]=0.55;

SCALAR FUNCTIONS

The logarythm , ln( ), exponential, exp( ), absolute value, abs( ), signal, sign( ), maximum element, maxe( ), minimum element, mine( ), and internal summation, sum( ) functions can be applied to scalars as well as to arrays. Like the arithmetic operations, these are pointwise functions when applied to arrays. For example:

lnER[a]=ln(er);

The maximum, max(,) and the minimum, min(,) functions can be applied to pairs of scalars or arrays. Again these functions are computed pointwise when they take array arguments. For example:

a1={1@AA, 2@BB, 3@DD}; # zero@CC is implicit

b1={5@BB, 1@DD};

c1=max(a1, b1);

# c1 is {1@AA, 5@BB, 0@CC, 3@DD};

CONSTRAINTS

The model's constraints establish the conditions to be satisfied by the selected portfolios. The first constraint, that must be always present, is a normalization condition:

normal: sum[all] $ == 1;

The normal constraint indicates the total value to be invested in the portfolio, which is usually taken as 1 or 100, so that the selected portfolio's composition appears as unit fractions or

percentages. $ are the model's decision variables representing the value to be invested in each asset in the universe. Equality constraints are indicated by the symbol "==", and inequality constraints are indicated by the symbols "<=" and ">=".

Important note:

1. Do not confuse the attribution, "=" with, the equality symbol, "==".
2. The equality constraints must be established <u>prior</u> to the inequality constraints.

We can define constraints on weighted sums of investments in a given domain, as in: *foodsbound: sum[food] $ >= 0.2;* establishing that sums of investments in the *food* sector shall not exceed 20%.

We can likewise establish sets of constraints as in:

ineqmin: $[TEL4]>=0.3;

upperbound: for[all] $ <=0.4;

*ineqmax: for[private] $ <= 0.1*x;*

establishing that investment in *TEL4* must be at least 30%, that each asset individually shall not exceed 40% of the portfolio, and that the investment in each asset of the *private* sector shall not exceed 10% of the value of array "x" at that asset.

The Markowitz model

The decision variables in a Markowitz type model are interpreted as the amount to be invested in each asset of the universe. The objective to be optimized is a utility function on the portfolio's expected return as compared to the variance of this return. The main inputs to the model are the expected returns, in array er, and the corresponding covariance matrix, S. The amount invested in each asset, say x, must be always positive. These are the signal constraints. Other equality and inequality constraints may be imposed in the form of Te*x==te and Tl*x<=tl. Thus the general form of the optimization problem is:

*max U(x) = h x'*er -x'*Q*x*

where

x is the amount invested in each asset (to be found)
er=E(return) is the expected returns array
S=Cov(return) is the returns' covariance matrix
Te is the equality constraints coefficient matrix
te is the equality constraints independent vector
Tl is the inequality constraints coefficient matrix
tl is the inequality constraints independent vector
h is the ponderation parameter between the portfolio's expected return and risk (variance). The larger h (this is the Greek letter eta), the higher is the investor's propension to accept risk for higher expected return. Smaller etas, quantify a higher risk aversion of the investor.

We are faced with a parametric quadratic programming optimization problem. The solution, for a given value of eta, is the viable vector x (i.e. a vector x satisfying all the model's constraints) optimizing the utility function. Thus, x@asseta is the amount invested at asseta in the optimal portfolio. Since for each value of eta we have a different (optimal) solution, we shall write the solution as x(h).

Each optimal (or efficient) solution represents a portfolio whose expected return and variance are *e(h)=x(h)'x*er* and *v(h)=x(h)'*S*x(h)*. The standard deviation is just the variance square root *s(h)=sqrt(v(h))*. The curve *e(h)* versus *s(h)* is the model's efficient frontier.

The efficient frontier is composed by quadratic segments that form a continuous and concave curve. It represents the top left border of the viable portfolios set. To trace it, we first have to compute all critical points. Each critical point corresponds to a critical value of eta. The critical etas are those in which a qualitative change in the portfolio occurs, i.e.:

1. an asset enters or leaves the portfolio, or
2. a constraint goes from slack (or inactive) to tight (or active), or vice versa.

In order to refer to a portfolio at a given position in the efficient frontier, it is costumary to use a statement like "the efficient portfolio of return e more or less s".

Application example

Let us take the example model of the section that explains the MODEL.CP file, with expected return and standard deviations obtained with the filter's default parameters.

Using the numerical template to select some points in the efficient frontier, we can make the following report:

Selected Portfolios: Parameters, Assets and Composition

Parameters of portfolio A

eta	*esp*	*var*	*std*	*rate*	*k*	*l*
4,22E-01	*9,39E-02*	*4,28E-02*	*2,07E-01*	*-7,51E-03*	*1*	*0,00E+00*

Assets and Composition of portfolio A

5,00E-01@PET4 2,00E-02@BBD4 3,00E-02@SCO4 5,00E-02@CEV4 4,00E-01@BRH4

Parameters of portfolio B

eta	*esp*	*var*	*std*	*rate*	*k*	*l*
3,21E-01	*8,72E-02*	*3,78E-02*	*1,95E-01*	*-3,05E-02*	*2*	*0,00E+00*

Assets and Composition of portfolio B

4,03E-01@PET4 2,00E-02@BBD4 3,00E-02@SCO4 5,00E-02@CEV4 4,97E-01@BRH4

...and so on...

Filter

The Markowitz Model requires as input data, usually supplied by the user, arrays with the assets' expected returns and standard deviations. Estimates of these parameters may be computed based on historical prices of each asset, or by any other means that the user deems convenient. A simple tool for this purpose, i.e. a filter, is part of the Critical-Point software.

The filter operates over long time series of prices for each asset under consideration. The installation disk contains some example time series for a few assets of the Brazilian stock market. Up-to-date series can be sought from several financial data distribution companies.

The parameters in the Option Filter menu are:

1. **Final date**	The final quotation date to be considered;
2. **Interval**	The interval between quotations dates to be considered;
3. **Samples**	How many samples to consider;
4. **Extrap**	How many intervals ahead of the final date to project he expected returns and standard deviations;
5. **Deflator**	Name of the file with the official quotation dates;
6. **Extension**	File name extension of the quotation files;
7. **Hurst**	Fractal characteristic parameter of the underlying Brownian motion (between 0.5 and 1).

The filter (F) computes the mean return and standard deviation for each asset in the historical time series. Price samples are taken in a date domain T, specified by the filter's parameters:

t in T = [final_date, final-date -interval, final_date -2*interval, ..., final_date -samples*interval];

The filter estimates logarithmic (or continuos) returns; writing price(i,t) and r(i,t) for the price and the return of asset i at time t, the filter computes

rl(i,t) = ln(price(i,t+1)/price(i,t));

Next, the filter computes the mean return of each asset,

meanl(i) = (1/samples) sum[T] rl(i,t);

and the covariance matrix, standard deviations and correlation matrix of the samples.

Sl(i,j) = (1/samples) sum[T] (rl(i,t) - meanl(i))*(rl(j,t) - meanl(j));

sl(i) = Sl(i,i)^(1/2);

Corrl(i,j) = Sl(i,j)/(sl(i)*sl(j));

Next, the filter extrapolates the logarithmic returns and standard deviations extrap intervals to the future:

erl(i) = extrap*meanl(i);

stdl(i) = extrap^Hurst*sl(i);

Finally, the filter converts all logarithmic mesures of return and risk to simple rates of return and risk. For deterministic prices at time t=0 and t=T , the logarithmic return is rl = ln(S(T)/S(0)), the simple rate of return is r = (S(T)-S(0))/S(0) , so r = exp(rl) -1 . The filter assumes the prices have a log-normal distribution, so the transformation rules become more involved, like

r = exp(erl +0.5 stdl^2) -1 , s^2 = exp(2r +sl^2)(exp(sl^2) -1) .

The filter saves the extrapolated mean rates of return and standard as associative arrays in the file FILTER.CP, and the correlation matrix in the matricial file CORRELF.M.

The universe set, i.e. all, used by the filter is the original universe set that also appears in file FILTER.CP. In the file MODEL.CP we can restrict the original universe set, i.e. we can remove some of its elements. When we restrict the original universe set, the MDL compiler extracts automatically from the er, std, and correll, only the subarrays and submatrices it needs. The professional version users of Critical-Point can receive via modem updates of these arrays and matrices that are transmitted for a large number of Brazilian assets. The filters used to compute these transmitted data are far more sophisticated than the filter in the Critical-Point software. These filters incorporate advanced techniques like fractal analysis, adaptive filters, neural networks, special treatment of short series and missing data, and so on.

DERIVATIVES

The Derivative tool expands the expected returns and standard deviation arrays, as well as the correlation matrix, to include derivative assets over object assets already in the fundamental set.

```
put={OTP19@TEL4, OTP24@TEL4 };

call={OTC16@TEL4, OTC17@TEL4 };

exdays={40@OTP19, 30@OTP24, 40@OTC16, 40@OTC17};

O={6.7@OTP19, 10.9@OTP24,10.9@OTC16, 6.7@OTC17};

S={63.2@OTP19, 63.2@OTP24, 63.2@OTC16, 63.2@OTC17};

K={64@OTP19, 72@OTP24, 56@OTC16, 68@OTC17};
```

In the file DERIV.CP example listed above, we have the set *put* declaring the (european) put options OTP19 and OTP24 over the underlying asset TEL4; similarly we declare call options OTC16 and OTC17. In order to define the options we shall specify each option's price or premium (O), the underlying asset price (S), the striking price (K), and remaining days for expiration (exdays).

DERIVATIVE COVARIANCES

The derivative tool writes the extended universe set (all), with the name of all assets (fundamental and derivatives), and its expected rates and standard deviations in file ERSX.CP, and the extended correlation matrix in CORRELX.M.

The simple simple rates of return of a call option is:

rc= (max(S(t1) -K(t0) , 0) -O(t0)) / O(t0)

The Expected returns, standard deviations and correlation for derivative assets are computed by numerical integration, assuming log-normality of the underlying asset price distribution.

(see J.M.Stern et all, Metodos de Otimizacao em Financas).

MATRICIAL FILES

The MDL compiler compiles, i.e. translates, the model described in MDL language to matrix form. The problem in matrix form, in the file MARKIN.M, is the input to the parametric quadratic programming solver (Q). To build more sophisticated models, we must understand the structure of all matrix files.

The order in which assets appear in positional vectors or matrices is the order in which they appear in the original universe set at file MODPS.CP. The filter has a procedure to compute the correlation matrix. The correlation matrix element **Cor(i,j)** is interpreted as a measure of mutual interference between the **i**-th and **j**-th asset returns. For example, if both assets have proportional returns **pi=a*pj**, **a>0**, then **Cor(i,j)=1**. If, on the other hand, the proportionality constant is negative, **a<0** (i.e. an asset i price increase implies an asset j price decrease), then **Cor(i,j)=-1**. Finally, if both returns are mutually independent, then **Cor(i,j)=0**.

A sophisticated user may specify the correlation matrix by himself . Critical-Point allows the direct manipulation of the matrix files. However, to give a matrix that does not satisfy all the mathematical properties of a correlation matrix may cause numerical errors in the parametric quadratic programming procedure (Q). Amongst all the properties of the correlation matrix, we have:

1- Diagonally Dominance :	**Cor(i,i)==1**; **abs(Cor(i,j))<1, i�j**;
2- Symetry :	**Cor(i,j)==Cor(j,i)**;
3- Positivity :	**min(eig(Cor))>0**;

HISTORICAL RETURNS versus MARKET ANALYSIS PREDICTIONS

After reading the files ERSX.CP and MODEL.CP, the MDL compiler reads file CORRELX.M in order to regenerate the covariance matrix.

S(i,j)=s(i)*Corr(i,j)*s(j);

Finally the MDL compiler saves all the constraints established on MODEL.CP, as well as the expected returns vector and the covariance matrix in the file MARKIN.M, which is the input for the parametric quadrametic programming procedure (Q).

The MDL compiler treats the expected returns and standard deviations arrays as any other associative array, so they can be modified using all the MDL language resources (to avoid inconsistencies with the derivatives computed expected rates and volatilities, these modifications should be done in file DERIV.CP). It is worth mentioning that professional market analysts almost never use extrapolated means as predicted returns. The expected returns and possibly also the standard deviations, extrapolated by the filter, are only used to "calibrate" the model, but these values are adjusted by the analysts. Among the most popular techniques to make these adjustments we have:

1- Fundamentalist analysis prediction
2- Graphic analysis prediction
3- Technical analysis forecast
4- Mathematical methods as neural nets, adaptive filters and classification trees.

These techniques do not belong to the scope of this manual, and are the object of specific disciplines at NOPEF -USP.

Data Files

The filter reads price time series from the DATA directory. File dolof.ofc has the American Dollar to Brazilian Real exchange rate; This file also defines the official calendar, i.e. the trading days. Other *.ofc files have dollar deflated price series for some Brazilian assets. The 3 first lines in each file define:

1- The Asset name,
2- The price deflator,
3- The number of shares.

The time series terminator is the asterisk symbol, * . The next example shows file TEL3.ofc, with dollar prices for 10000 shares of TELebras, the Brazilian Telecommunications Company, at the Sao Paulo Stock Exchange, BOVESPA, in the last 4 calendar days of 1994. Dates must be in dd/mm/yyyy format, and prices in fixed or floating point format.

Asset: TEL3

Deflator: DOLOF.OFC

Shares: 1E+4

Date Price

30/12/1994 4.314E+02

29/12/1994 4.304E+02

28/12/1994 4.030E+02

27/12/1994 4.025E+02

* (T.S. terminator)

(End of File)

Logical Restrictions over Sets

The MDL language allows us to restrict sets by logical expressions. For example:

exampleset[energy|bank]=(arrbank>0.5)|(arrenergy<0.2);

after this statement, only the assets in the *bank* sector that also satisfy the condition *arrbank>0.5* or the assets in the *energy* sector that satisfy the condition *arrenergy<0.2* remain in set *exampleset*.

Logical conditions can be built using the comparison operators:

> more than
<= more or equal than
< less than
<= less or equal than
== equal to
~= not equal.

More complex operations can be done between sets and arrays. The representation of a set is actually nothing else than a boolean-valued (0 or 1) associative array. Thus, in attributions to sets we must interpret the name of an asset, asseta, as equivalent to the expression 1@asseta. We also interpret logical operations between sets using the logical operators not, ~, and, &, and or, |, as taking the value 0 as false and any non zero value as true.

When we restrict the universe set, all, we forbid any excluded asset to enter the portfolio, and only the necessary subvectors and submatrices of p, std, and cov will be written by the MDL compiler into the MARKIN.M file. To keep the model consistent, we must first restrict the universe set in MODEL.CP , and only then establish the model's constraints.

Depuration (Debugging)

Even the most experienced programmers make errors and, although very few financial models require all the resources of the MDL language, some models may become complex. The MDL compiler has some simple tools to help you find programming errors (bugs). When the compiler cannot analyze (parse) a statement, it exhibits an error message. This message

contains:

1 - The name of the file where the error was detected.
2 - The line number where the error is.
3 - The last valid token parsed, the last character read, and its ASCII code.

The command print x; forces the MDL compiler to write the object x (scalar, set, or array) to the file PRINT.LOG, exactly as the object is defined at the moment the print command is issued in the MDL program.

The Mean-Variance Calculator

The mean-variance calculator (C button) provides an easy way for evaluating the expected return and standard deviation of pre-defined portfolios. The portfolios are defined as arrays in file FOLIOS.CP , for example:

papas = { 5.0E-1@PET4, 2.0E-2@BBD4, 3.0E-2@SCO4, 5.0E-2@CEV4, 4.0E-0@BRH4 };

The calculator prints the computed values in file RFOLIOS.CP, for example:

papas: e = 9,39E-02 s = 2,07E-01

Short Selling

In a model description, before the constraints declarations, the *short* subset (of the universe) can be declared. The effect of having an asset in *short* is to change the sign of its expected return, as well as to change the sign of its correlation with all other assets <u>not</u> in *short.* The correlation of an asset in *short* with an other asset <u>also</u> in *short*, as well as its volatility, is unchanged.

The financial interpretation of having an asset in *short* is to sell the asset short in the sense of LIntner: A short sale involves the deposit of a cash collateral, equal to the amount sold short. Therefore a short sale must not be seen as a source of money, but rather a use of it, to be included in the *normal* constraint like any other investment in the portfolio.

If the investor receives an interest on the collateral, the expected return array must be adjusted, what can easily be accomplished using the *short* set as a domain for array operations. In the next example we consider portfolios allowing some long and some short positions. The collateral for any sort sale is to receive a 1% fixed interest.

all={ TEL4, ELE6, PET4, BB4, BBD4, SCO4, CEV4, BRH4, IBOVESPA };

short={TEL4,ELE6};

p[short]=p -0.01; #adding 1% to -p

normal: sum[all] $==1;

Tracking

In many applications it is desirable to form a portfolio to follow, or track, a given reference investment. The difference in returns between the portfolio and the reference investment is the tracking error. In the next example, we look for efficient portfolios to track the asset IBOVESPA (a standard Brazilian index, much like SP500). The mean-variance depicted in the efficient frontier for this model is no longer that of absolute returns, but of returns relative to the reference investment (IBOVESPA).

all={ TEL4, ELE6, PET4, BB4, BBD4, SCO4, CEV4, BRH4, IBOVESPA };

short={IBOVESPA};

normal: sum[all] $==2;

track: for[short] $==1;

Who wrote Critical-Point

Julio Michael Stern wrote the original version of the solver, in Matlab, and specified the MDL language and the user interface. Jacob Zimbarg Sobrinho ported the solver to C, using Watcom C, and the wrote the filter. Fabio Nakano wrote the MDL compiler, using GNU lex and yacc (flex and bison), and the current version of the user interface, in Visual Basic. Cibele Dunder wrote derivative tool.

Bugs and Suggestions

Please report bugs and suggestions to jstern@ime.usp.br (internet).