MVN *package*

Empirical Market Microstructure

(2006, Oxford University Press)

Companion *Mathematica* notebook

Joel Hasbrouck

This package defines a multivariate normal distribution "object" (i.e, a representation for variables that are multivariate normal), and various routines to work with them. The notebook should be stored as a package.

■ Properties of the multivariate normal distribution:

The package implements functionality based on the following relations.

Consider a multivariate normal distribution:

$$\underset{n\times 1}{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix};\ E[X] = \mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix};\ Var[X] = \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \tag{1}$$

The partitions here must be consistent, but are otherwise arbitrary. Either X_1 or X_2 or both may be scalars.

If $X \sim MVN(\mu, \Sigma)$, then

$$E[X_1 \mid X_2] = \mu_1 + (X_2 - \mu_2)\,\Sigma_{22}^{-1}\,\Sigma_{21} \tag{2}$$

$$Var[X_1 \mid X_2] = \Sigma_{11} - \Sigma_{12}\,\Sigma_{22}^{-1}\,\Sigma_{21} \tag{3}$$

Consider a linear transform $Y = AX + b$, where A is a rectangular matrix and b is a column vector. Then:

$$Y \sim MVN(A\mu + b, A\Sigma A') \tag{4}$$

```
BeginPackage["MVN`"];
```

▱ *The MVN "object"*

```
MVN::usage =
  "MVN[μ,Σ,v] is an object that represents a multivariate normal distribution
    with mean vector μ and covariance matrix Σ. The third parameter, v,
    is list of Mathematica expressions that correspond to the variables.
    (Typically they are variable names or formulas for the variables.)";
```

```
GetMean::usage =
  "GetMean[mvn] where mvn is an MVN object returns the mean vector of mvn.";
```

```
SetMean::usage = "SetMean[mvn,μNew] where mvn is an
    MVN object returns mvn with the mean vector set to μNew.";
```

```
GetVariance::usage = "GetVariance[mvn] where
    mvn is an MVN object returns the variance matrix of mvn.";
```

```
SetVariance::usage =
  "SetVariance[mvn,ΣNew] where mvn is an MVN object returns mvn
    with the variance matrix set to ΣNew.";
```

```
GetLabel::usage =
  "GetLabel[mvn] where mvn is an MVN object returns the label vector of mvn.";
```

```
SetLabel::usage =
  "SetLabel[MVN,v2] returns an MVN object in which the vector of
    variable expressions is replaced by v2.";
```

```
StandardMVN::usage =
  "StandardMVN[n,x] returns an MVN of order n with zero mean and
    a covariance matrix equal to the identity matrix. The
    variables are labeled x1,...,xn. If the second argument
    is omitted, the variables are labeled z1,...,zn";
```

```
LinearForm::usage =
  "LinearForm[mvn1,b,c] (where b is a matrix and c is a vector) returns
    an MVN that describes a linear transformation of mvn1. If
    mvn1=MVN[μ,Σ,v] then the return MVN is the distribution of b.v+c.
    LinearForm[mvn1,b] returns an MVN giving the distribution of b.v";
```

```
MakeLinearForm::usage =
  "MakeLinearForm[MVN0,b] returns an MVN object representing the
    distribution of a set of linear combinations of the variables
    in MVN0=MVN[μ,Σ,v]. b is a list of expressions defining
    linear combinations of v (and a constant, if desired).";
```

```
MVNConditional::usage =
  "MVNConditional[MVN0,v1,v2] returns an MVN object representing a
    conditional multivariate normal distribution. MVN0=MVN0[μ,Σ,v] is
    the joint distribution. v2 is a list of the conditioning variables;
    v1 is a list of the target variables. Both v1 and v2 must map to
    (correspond to) v. MVNConditional[MVN0,v1] returns the conditional
    distribution of v1 conditioned on all variables that are not in v1.";
```

```
Format[MVN[μLocal_, ΣLocal_, vLocal_]] := RowBox[
   {mpForm[vLocal], " ~ ", StyleBox[𝒩, FontSize → 16, FontWeight → Bold], "(",
    mpForm[μLocal], ",", mpForm[ΣLocal], ")"}] // DisplayForm
```

```
Begin["`Private`"];
```

▢ *DisplayForm formatting of MVN objects*

— *General::spell : Possible spelling error: new symbol name "Bold" is similar to existing symbols {Fold, Hold}. More…*

▢ *Getting and setting parameters*

```
GetMean[MVN[μ_, Σ_, v_]] := GetElement[μ];
```

```
SetMean[MVN[μ_, Σ_, v_], μNew_] := MVN[μNew, Σ, v]
```

— *General::spell1 : Possible spelling error: new symbol name "SetMean" is similar to existing symbol "GetMean". More…*

```
GetVariance[MVN[μ_, Σ_, v_]] := GetElement[Σ]
```

```
SetVariance[MVN[μ_, Σ_, v_], ΣNew_] := MVN[μ, ΣNew, v]
```

— *General::spell1 : Possible spelling error: new symbol name "SetVariance" is similar to existing symbol "GetVariance". More…*

```
GetLabel[MVN[μ_, Σ_, v_]] := GetElement[v];
```

```
SetLabel[MVN[μ_, Σ_, v_], v2_] := MVN[μ, Σ, v2]
```

— *General::spell1 : Possible spelling error: new symbol name "SetLabel" is similar to existing symbol "GetLabel". More…*

▢ *Constructing and transforming MVN's*

```
StandardMVN[1, v_ : Global`z] := MVN[0, 1, v]
```

```
StandardMVN[n_, v_ : Global`z] :=
 MVN[Table[0, {n}], IdentityMatrix[n], Table[v_i, {i, 1, n}]]
```

```
LinearForm[MVN[μ_, Σ_, v_], b_, c_ : 0] :=
 MVN[b.ToVector[μ] + c, b.ToMatrix[Σ].Transpose[b], b.ToVector[v] + c]
```

```
MakeLinearForm[MVN[μ_, Σ_, v_], vNew_] := Module[{b, b0},
  b = Transpose[Coefficient[ToVector[vNew], #, 1] & /@ ToVector[v]];
  b0 = ToVector[vNew] - b.ToVector[v];
  LinearForm[MVN[μ, Σ, v], b, b0]
 ]
```

□ *Conditional distributions*

```
MVNConditional[MVN[μ_, Σ_, v_], v1a_, v2a_: 0] := Module[{v1, v2, i1, i2},
  v1 = ToVector[v1a];
  i1 = Position[v, #] & /@ v1 // Flatten;
  i2 =
   If[v2a =!= 0,
        v2 = ToVector[v2a]; Position[v, #] & /@ v2 // Flatten,
        Complement[Range[Length[ToVector[v]]], i1]
   ];
  MVN[
   μ[[i1]] + (v[[i2]] - μ[[i2]]).Inverse[Σ[[i2, i2]]].Σ[[i2, i1]],
   Σ[[i1, i1]] - Σ[[i1, i2]].Inverse[Σ[[i2, i2]]].Σ[[i2, i1]],
   v[[i1]]]
 ]
```

— *General::spell1 : Possible spelling error: new symbol name "vRange" is similar to existing symbol "Range".* More…

□ *Helper functions*

```
mpForm[x_] := Which[MatrixQ[x] && ((Times @@ Dimensions @ x) == 1), x[[1, 1]],
   VectorQ[x] && (Length[x] == 1), x[[1]], True, MatrixForm[x]];
```

```
GetElement[x_] := Which[MatrixQ[x] && ((Times @@ Dimensions @ x) == 1), x[[1, 1]],
   VectorQ[x] && (Length[x] == 1), x[[1]], True, x];
```

```
ToVector[x_] := Flatten[{x}];
```

```
ToMatrix[x_] := Which[MatrixQ[x], x, VectorQ[x], {x}, True, {{x}}];
```

```
End[];
```

```
EndPackage[]
```