ESBMaths

Units

ESBMaths
ESBMaths2

Types

TBitList
TDynFloatArray
TDynFloatMatrix
TDynLIntArray
TDynLIntMatrix
TDynLWordArray
TDynLWordMatrix

Routines

GrandMean
HarmonicMean
IGreatestPowerOf2
ILog2
IncLim
IncLimI
IncLimL
IncLimSI
IncLimW
IncompleteBeta
IntPow
InverseAll
InverseGamma
IsPositiveEArray
ISqrt
LCM
LinearTransform
LnAll
LnGamma
Log10All
LogXtoBaseY
MatricesSameDimensions
MatrixDimensions
MatrixIsRectangular
MatrixIsSquare
Max3Word
Max4Word
MaxB
MaxBArray
MaxCArray
MaxEArray
MaxExt
MaxI
MaxIArray
MaxL
MaxLArray
MaxSArray
MaxSI
MaxSIArray
MaxW
MaxWArray
Min3Word
Min4Word
MinB
MinBArray
MinCArray
MinEArray
MinExt
MinI
MinIArray
MinL
MinLArray
MinSArray
MinSI
MinSIArray

MinW
MinWArray
MultiplyMatrices
MultiplyMatrixByConst
MultiplyMatrixByConst2
MultVectors
Norm
PermutationX
Polar2XY
PopulationVariance
PopulationVarianceAndMean
Pow2
RelativePrime
SameFloat
SampleVariance
SampleVarianceAndMean
Set87ControlWord
Sgn
Sign
SquareAll
SubtractFromMatrix
SubtractMatrices
SubVectors
SumBArray
SumBArray2
SumCArray
SumEArray
SumIArray
SumLArray
SumLWArray
SumSArray
SumSIArray
SumSIArray2
SumSqDiffEArray
SumSqEArray
SumWArray
SumWArray2
SumXYEArray
SwapB
SwapC
SwapDbl
SwapExt
SwapI
SwapI32
SwapInt64
SwapL
SwapSI
SwapSing
SwapW
TenToY
TransposeMatrix
TwoToY
UMul
UMulDiv
UMulDiv2p32

UMulMod
XtoY
XY2Polar

Global Variables

ESBTolerance

Global Constants

Cbrt10
Cbrt100
Cbrt2
Cbrt3
CbrtPi
ESBe
ESBe2
ESBePi
ESBePiOn2
ESBePiOn4
ESBGamma
ESBPi
FourPiOn3
InvCbrtPi
InvPi
InvSqrt2
InvSqrt3
InvSqrt5
InvSqrtPi
Ln10
Ln2
LnPi
LnRt2Pi
Log10Base2
Log2Base10
Log3Base10
LogEBase10
LogPiBase10
MaxCurrency
MaxDouble
MaxExtended
MaxSingle
MinCurrency
MinDouble
MinExtended
MinSingle
OneDegree
OneMinute
OneRadian
OneSecond
Pi2
PiOn2
PiOn3
PiOn4
PiToE
Sqrt10
Sqrt2
Sqrt3
Sqrt5
SqrtPi
ThreePi
ThreePiOn2
TwoPi

TwoToPower63

Routine Categories

Description
Routine Categories

Extra String Handling Routines

Various routines for string handling that supplement those found in SysUtils.

String/Integer Conversion Routines

Various routines for converting Integers into Strings, and Strings into Integers.

String/Float Conversion Routines

Various routines for converting Floats into Strings, and Strings into Floats.

Boolean Conversion Routines

Various routines for converting Booleans into Characters and Strings, and Characters into Booleans.

Complex Number Conversion Routines

Various routines for creating Complex Numbers and converting Complex Numbers into Strings or their components.

Complex Number Arithmetic Routines

Various routines for Mathematical Manipulation of Complex Numbers.

Comparison of Complex Numbers

Various Routines for comparing Complex Numbers. All Routines result in a Boolean.

Temperature Conversion Routines

Various Routines for Converting between units of Temperature.

Distance Conversion Routines

Various routines for Converting between different units of Distance.

Mass Conversion Routines

Various routines for Converting between different units of Mass.

Volume Conversion Routines

Various routines for Converting between different units of Volume.

Area Conversion Routines

Various routines for Converting between different units of Area.

Pressure Conversion Routines

Various routines for Converting between different units of Pressure.

Velocity Conversion Routines

Various routines for Converting between different units of Velocity.

Acceleration Conversion Routines

Various routines for Converting between different units of Acceleration.

Force Conversion Routines

Various routines for Converting between different units of Force.

Energy Conversion Routines

Various routines for Converting between different units of Energy.

Power Conversion Routines

Various routines for Converting between different units of Power.

Fuel Consumption Conversion Routines

Various routines for Converting between different units of Fuel Consumption.

Flow Conversion Routines

Various routines for Converting between different units of Flow.

Torque Conversion Routines

Various routines for Converting between different units of Torque.

Currency Conversion Routines

Various routines for Converting between Currencies.

Date/Time Conversion Routines

Various routines for Converting between TDateTime andand Time Portions.

Date/Time Arithmetic Routines

Various routines for Manipulating Dates and Times.

Week Based Arithmetic Routines

Various routines for Manipulating Week based Info.

Year Based Arithmetic Routines

Various routines for Manipulating Year based Info.

Month Based Arithmetic Routines

Various routines for Manipulating Month based Info.

Date/Time Comparison

Routines to compare Dates and Time. All routines return a Boolean Value.

Financial Conversion Routines

Various routines for Creating and Converting between various Financial Types and Strings, Floats etc.

Financial Arithmetic Routines

Various routines for Manipulating various Financial types.

Fraction Conversion Routines

Convert Between Fractions and Floats, Strings, etc.

Fraction Arithmetic Routines

Various routines for arithmetic manipulation of Fractions

Fraction Comparison

Various routines for Comparing Fractions. All Routines result in a Boolean.

Area Calculation Routines

Various routines for computing Perimeters, Areas and Surface Areas.

Volume Calculation Routines

Various routines for computing Volumes.

Mixed Imperial Conversion Routines

Various routines for Creating and Converting between various Mixed Imperials and Floats.

Arithmetic Routines for Mixed Imperials

Various routines for Manipulating Mixed Imperial types.

Conversions between Integers/Floats and Strings

Various routines for Converting between Integers/Floats and Strings.

Comparison between Integers and Floats

Various routines for Comparison between Integers and Comparison between Floats. All Routines result in a Boolean.

Arithmetic Routines for Floats

Various Routines for Manipulating Floating Point types

Arithmetic Routines for Integers

Various routines for Manipulating Integer types.

Arithmetic Routines for Matrices

Various Routines for Manipulating the contents of Matrices

Comparision Routines for Matrices

Various Comparison routines for Matrices. All Routines result in a Boolean.

Conversion Routines for Matrices

Various routines for Creating and Converting Matrices.

Arithmetic Routines for Vectors

Various routines for Creating and converting Vectors.

Vector Comparison Routines

Various Routines for comparing vectors. All routines return Boolean.

Conversion Routines for Vectors

Various routines for Creating and Converting Vectors.

Statistical Routines for Probability and Distributions

Various Routines for Probability calculations and for Manipulating Probability Distributions.

Descriptive Statistical Routines

Various routines for calculating Descriptive Statistics.

Statistical Routines for Regression

Various routines for Computing Regressions, Lines of Best Fit and associated values.

Physics Routines

Various routines for Mechanics and other Physics-based formulae.

Routines that produce Dialogs

Various routines for Creating and Using Dialogs.

System Operations

Various routines for accessing and utilising the Operating System functions.

Memory Operations

Various Routines for low level manipulations of Bits, Bytes through to blocks of Memory.

Resource Management Routines

Various routines for Managing Resources.

ESB System Routines

Various routines for getting and setting ESBPCS Information from the registry.

Country Based Routines

Various routines for nadling country based information, that is stored in XML.

String Handling Routines

Various utility routines/method/classes for string handling

Description
No description yet...

File/Directory Name Manipulation Routines

Various Routines for Manipulating Files and Directories

Components

Buttons

Labels

XPath Related Stuff

Units etc related to DLogikk's implementation of XPath.

Legend

 - Marks that the item has an associated example. (this bitmap is a hyperlink.)
 - Marks that the item has documented bugs.
 - Marks that the item has documented todo's.
(A todo is something which should be fixed before the next release (or "real soon"!))

Relevant to classes and interfaces only

 - Marks that the class/interface has a property, method or event with examples.
 - Marks that the class/interface has a property, method or event with documented bugs.
 - Marks that the class/interface has a property, method or event with documented todo's.

Note that a symbol in the last group is not present if the corresponding symbol in the first group is present.

Welcome to MyLib

...Write some nice words here...

Introduction

...write text here...

ESBMaths2 Unit {button &Top,JI(`',`IDH_Unit_ESBMaths2')}{button &Types,JI(`',`IDH_UnitTopic_ESBMaths2_OtherTypes')} {button &Routines,JI(`',`IDH_UnitTopic_ESBMaths2_Routines')}

Dependencies Legend

ESBMaths 3.2.1 - contains useful Mathematical routines for Delphi 4, 5 & 6.

Description

These routines are used by ESB Consultancy within the development of their Customised Applications, and have been under Development since the early Turbo Pascal days. Many of the routines were developed for specific needs.

We does ask that if this code helps you in you development that you send as an email mailto:glenn@esbconsult.com.au or even a local postcard. It would also be nice if you gave us a mention in your About Box or Help File.

ESB Consultancy Home Page: http://www.esbconsult.com.au

Mail Address: PO Box 2259, Boulder, WA 6449 AUSTRALIA

Check out our new ESB Professional Computation Suite with 3000+ Routines and 80+ Components for Delphi 4, 5 & 6.

http://www.esbconsult.com.au/esbpcs.html

Also check out Marcel Martin's HIT at:

http://www.esbconsult.com.au/esbpcs-hit.html

Marcel has been helping out to optimise and improve routines.

Rory Daulton has generously donated and helped with many optimised routines. Our thanks to him as well.

Marcel van Brakel has also been very helpful has includes ESBMaths into the Jedi Collection. http://www.delphi-jedi.org/ Any mistakes made are mine rather than Rory's or the Marcels'.

History: See Whatsnew.txt

Other Types

TDynFloatArray
TDynFloatMatrix
TDynLIntArray
TDynLIntMatrix
TDynLWordArray
TDynLWordMatrix

Routines

AddMatrices
AddToMatrix
AddVectors
DotProduct
GrandMean
InverseAll
LinearTransform

LnAll
Log10All
MatricesSameDimensions
MatrixDimensions
MatrixIsRectangular
MatrixIsSquare
MultiplyMatrices
MultiplyMatrixByConst
MultiplyMatrixByConst2
MultVectors
Norm
SquareAll
SubtractFromMatrix
SubtractMatrices
SubVectors
TransposeMatrix

AddMatrices Function

Unit
ESBMaths2

Declaration
```
Function AddMatrices(const X, Y: TDynFloatMatrix): TDynFloatMatrix;
```

Description
Both X and Y must be truly Rectangular and must be of the same dimension otherwise an Exception is raised.

Implementation

```
function AddMatrices (const X, Y: TDynFloatMatrix): TDynFloatMatrix;
var
   I, J, N: Integer;
begin
   Result := nil;
   if (High (X) < 0) or (High (Y) < 0) then
      raise EMathError.Create ('Matrix is Empty!');

   if (High (X) <> High (Y)) then
      raise EMathError.Create ('Matrices must be the same Dimension to Add!');

   N := High (X [0]);
   SetLength (Result, High (X) + 1, N + 1);
   for I := 0 to High (X) do
   begin
      if (High (X [I]) <> N) then
      begin
         Result := nil;
         raise EMathError.Create ('Matrices must be truly rectangular to
Add!');
      end;
      if (High (Y [I]) <> N) then
      begin
         Result := nil;
         raise EMathError.Create ('Matrices must be the same Dimension to
Add!');
      end;

      for J := 0 to N do
         Result [I, J] := X [I, J] + Y [I, J];
   end;
End;
```

AddToMatrix Procedure

Unit
ESBMaths2

Declaration
```
Procedure AddToMatrix(var X: TDynFloatMatrix; const Y: TDynFloatMatrix);
```

Description
Add one Matrix to another, X := X + Y. Both X and Y must be truly Rectangular and must be of the same dimension otherwise an Exception is raised.

Implementation

```
procedure AddToMatrix (var X: TDynFloatMatrix; const Y: TDYnFloatMatrix);
var
   I, J: LongWord;
     Rows, Columns: LongWord;
begin
   Rows := Length (X);
   Columns := Length (X [0]);
   if (Rows = 0) or (Columns = 0) then
      raise EMathError.Create ('Matrix is Empty!');

   if not MatricesSameDimensions (X, Y) then
      raise EMathError.Create ('Matrices must be the same Dimension to Add!');

   for I := 0 to Rows - 1 do
      for J := 0 to Columns - 1 do
         X [I, J] := X [I, J] + Y [I, J];
End;
```

AddVectors Function

Unit
ESBMaths2

Declaration
```
Function AddVectors(const X, Y: TDynFloatArray): TDynFloatArray;
```

Description
The Length of the resultant vector is that of the smaller of X and Y.

Implementation

```
function AddVectors (const X, Y: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, MinL (High (X), High (Y)) + 1);
   for I := 0 to High (Result) do
      Result [I] := X [I] + Y [I];
End;
```

DotProduct Function

Unit
ESBMaths2

Declaration
```
Function DotProduct(const X, Y: TDynFloatArray): Extended;
```

Description
the sum of the pairwise products of the elements. If Vectors are not of equal length then only the shorter length is used.

Implementation

```
function DotProduct (const X, Y: TDynFloatArray): Extended;
var
   I, N: Longword;
begin
   Result := 0.0;
   N := MinL (High (X), High (Y));
   for I := 0 to N do
      Result := Result + X [I] * Y [I];
End;
```

GrandMean Function

Unit
ESBMaths2

Declaration
```
Function GrandMean(const X: TDynFloatMatrix; var N: LongWord): Extended;
```

Description
Will handle non-Rectangular Matrices. Also returns N the number of Values since the Matrix may not be Rectangular

Implementation

```
function GrandMean (const X: TDynFloatMatrix; var N: LongWord): Extended;
var
   I, J: Integer;
begin
   Result := 0;
   if (High (X) < 0) or (High (X [0]) < 0) then
      raise EMathError.Create ('Matrix is Empty!');

   N := 0;
   for I := 0 to High (X) do
   begin
      N := N + Longword (High (X [I])) + 1;
      for J := 0 to High (X [I]) do
         Result := Result + X [I, J];
   end;
   if N > 0 then
      Result := Result / N
   else
      raise EMathError.Create ('Matrix is Empty!');
End;
```

InverseAll Function

Unit
ESBMaths2

Declaration
```
Function InverseAll(const X: TDynFloatArray): TDynFloatArray;
```

Description
An exception is raised if any element is zero

Implementation

```
function InverseAll (const X: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
   begin
      if X [I] = 0 then
         raise EMathError.Create ('Inverse of Zero');
      Result [I] := 1 / (X [I]);
   end;
End;
```

LinearTransform Function

Unit
ESBMaths2

Declaration
```
Function LinearTransform(const X: TDynFloatArray; Offset, Scale: Extended):
TDynFloatArray;
```

Description
NewX [i] = Offset + Scale * X [i]

Implementation

```
function LinearTransform (const X: TDynFloatArray;
   Offset, Scale: Extended): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
      Result [I] := OffSet + Scale * X [I];
End;
```

LnAll Function

Unit
ESBMaths2

Declaration
```
Function LnAll(const X: TDynFloatArray): TDynFloatArray;
```

Description
An exception is raised if any element is not Positive

Implementation

```
function LnAll (const X: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
   begin
      if X [I] <= 0 then
         raise EMathError.Create ('Logarithm on non-Positive');
      Result [I] := Ln (X [I]);
   end;
End;
```

Log10All Function

Unit
ESBMaths2

Declaration
```
Function Log10All(const X: TDynFloatArray): TDynFloatArray;
```

Description
An exception is raised if any element is not Positive

Implementation

```
function Log10All (const X: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
   begin
      if X [I] <= 0 then
         raise EMathError.Create ('Logarithm on non-Positive');
      Result [I] := ESBLog10 (X [I]);
   end;
End;
```

MatricesSameDimensions Function

Unit
ESBMaths2

Declaration
```
Function MatricesSameDimensions(const X, Y: TDynFloatMatrix): Boolean;
```

Implementation
```
function MatricesSameDimensions (const X, Y: TDynFloatMatrix): Boolean;
var
   M1, N1: LongWord;
   Rectangular1: Boolean;
   M2, N2: LongWord;
   Rectangular2: Boolean;
begin
   MatrixDimensions (X, M1, N1, Rectangular1);
   MatrixDimensions (Y, M2, N2, Rectangular2);
   Result := Rectangular1 and Rectangular2 and (M1 = M2) and (N1 = N2);
End;
```

MatrixDimensions Procedure

Unit
ESBMaths2

Declaration
```
Procedure MatrixDimensions(const X: TDynFloatMatrix; var Rows, Columns :
LongWord; var Rectangular: Boolean);
```

Description
Rows and Columns are the dimensions which really only make sense if the Matrix is Rectangular

Implementation

```
procedure MatrixDimensions (const X: TDynFloatMatrix;
   var Rows, Columns: LongWord; var Rectangular: Boolean);
var
   I: LongWord;
begin
   Rows := Length (X);
     if Rows > 0 then
      Columns := Length (X [0])
     else
        Columns := 0;

   Rectangular := False;
   if (Rows = 0) or (Columns = 0) then
      Exit;

   for I := 0 to Rows - 1 do
      if (LongWord (Length (X [I])) <> Columns) then
      begin
         Columns := 0;
         Exit;
      end;

   Rectangular := True;
End;
```

MatrixIsRectangular Function

Unit
ESBMaths2

Declaration
```
Function MatrixIsRectangular(const X: TDynFloatMatrix): Boolean;
```

Implementation
```
function MatrixIsRectangular (const X: TDynFloatMatrix): Boolean;
var
   I, N: Integer;
begin
   Result := False;
   if (High (X) < 0) then
      Exit;

   N := High (X [0]);
   for I := 0 to High (X) do
      if (High (X [I]) <> N) then
         Exit;

   Result := True;
End;
```

MatrixIsSquare Function

Unit
ESBMaths2

Declaration
```
Function MatrixIsSquare(const X: TDynFloatMatrix): Boolean;
```

Implementation
```
function MatrixIsSquare (const X: TDynFloatMatrix): Boolean;
var
   M, N: LongWord;
   Rectangular: Boolean;
begin
   MatrixDimensions (X, M, N, Rectangular);
   Result := Rectangular and (M = N);
End;
```

MultiplyMatrices Function

Unit
ESBMaths2

Declaration
```
Function MultiplyMatrices(const X, Y: TDynFloatMatrix): TDynFloatMatrix;
```

Description
The number of columns in X must equal the number of rows in Y

Implementation

```
function MultiplyMatrices (const X, Y: TDynFloatMatrix): TDynFloatMatrix;
var
   I, J, K: LongWord;
     XRows, XColumns: LongWord;
     YRows, YColumns: LongWord;
     XRectangular, YRectangular: Boolean;
begin
   Result := nil;
   MatrixDimensions (X, XRows, XColumns, XRectangular);
   MatrixDimensions (Y, YRows, YColumns, YRectangular);

   if not XRectangular or not YRectangular then
      raise EMathError.Create ('Matrices must both be Rectangular');
   if (XRows = 0) or (YRows = 0) then
      raise EMathError.Create ('Matrix is Empty!');
     if XColumns <> YRows then
      raise EMathError.Create ('Number of Columns in X does not equal'
             + #13 + 'the Number of Rows in Y');

   SetLength (Result, XRows, YColumns);

   for I := 0 to XRows - 1 do
      for J := 0 to YColumns - 1 do
          begin
             Result [I, J] := 0;
               for K := 0 to XColumns - 1 do
            Result [I, J] := Result [I, J] + X [I, K] * Y [K, J];
          end;
End;
```

MultiplyMatrixByConst Function

Unit
ESBMaths2

Declaration
```
Function MultiplyMatrixByConst(const X: TDynFloatMatrix; const K: Extended):
TDynFloatMatrix;
```

Description
Will handle non-Rectangular Matrices

Implementation

```
function MultiplyMatrixByConst (const X: TDynFloatMatrix; const K: Extended):
TDynFloatMatrix;
var
   I, J: Integer;
begin
   Result := nil;
   if (High (X) < 0) then
      raise EMathError.Create ('Matrix is Empty!');

   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
   begin
      SetLength (Result [I], High (X [I]) + 1);
      for J := 0 to High (X [I]) do
         Result [I, J] := X [I, J] * K;
   end;
End;
```

MultiplyMatrixByConst2 Procedure

Unit
ESBMaths2

Declaration
```
Procedure MultiplyMatrixByConst2(var X: TDynFloatMatrix; const K: Extended);
```

Description
Will handle non-Rectangular Matrices

Implementation

```
procedure MultiplyMatrixByConst2 (var X: TDynFloatMatrix; const K: Extended);
overload;
var
   I, J: LongWord;
     Rows, Columns: LongWord;
begin
   Rows := Length (X);
   if (Rows = 0) then
      raise EMathError.Create ('Matrix is Empty!');

   for I := 0 to Rows - 1 do
   begin
      Columns := Length (X [0]);
      for J := 0 to Columns - 1 do
         X [I, J] := X [I, J] * K;
   end;
End;
```

MultVectors Function

Unit
ESBMaths2

Declaration
```
Function MultVectors(const X, Y: TDynFloatArray): TDynFloatArray;
```

Description
The Length of the resultant vector is that of the smaller of X and Y.

Implementation

```
function MultVectors (const X, Y: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, MinL (High (X), High (Y)) + 1);
   for I := 0 to High (Result) do
      Result [I] := X [I] * Y [I];
End;
```

Norm Function

Unit
ESBMaths2

Declaration
```
Function Norm(const X: TDynFloatArray): Extended;
```

Description
the square root of the sum of the squares of the elements.

Implementation

```
function Norm (const X: TDynFloatArray): Extended;
begin
   Result := Sqrt (DotProduct (X, X));
End;
```

SquareAll Function

Unit
ESBMaths2

Declaration
```
Function SquareAll(const X: TDynFloatArray): TDynFloatArray;
```

Implementation
```
function SquareAll (const X: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, High (X) + 1);
   for I := 0 to High (X) do
      Result [I] := Sqr (X [I]);
End;
```

SubtractFromMatrix Procedure

Unit
ESBMaths2

Declaration
```
Procedure SubtractFromMatrix(var X: TDynFloatMatrix; const Y:
TDynFloatMatrix);
```

Description
Subtract one Matrix to another, X := X - Y. Both X and Y must be truly Rectangular and must be of the same dimension otherwise an Exception is raised.

Implementation

```
procedure SubtractFromMatrix (var X: TDynFloatMatrix; const Y:
TDynFloatMatrix);
var
   I, J: LongWord;
     Rows, Columns: LongWord;
begin
   Rows := Length (X);
   Columns := Length (X [0]);
   if (Rows = 0) or (Columns = 0) then
      raise EMathError.Create ('Matrix is Empty!');

   if not MatricesSameDimensions (X, Y) then
      raise EMathError.Create ('Matrices must be the same Dimension to Add!');

   for I := 0 to Rows - 1 do
   begin
      for J := 0 to Columns - 1 do
         X [I, J] := X [I, J] - Y [I, J];
   end;
End;
```

SubtractMatrices Function

Unit
ESBMaths2

Declaration
```
Function SubtractMatrices(const X, Y: TDynFloatMatrix): TDynFloatMatrix;
```

Description
Both X and Y must be truly Rectangular and must be of the same dimension otherwise an Exception is raised.

Implementation

```
function SubtractMatrices (const X, Y: TDynFloatMatrix): TDynFloatMatrix;
var
   I, J, N: Integer;
begin
   Result := nil;
   if (High (X) < 0) or (High (Y) < 0) then
      raise EMathError.Create ('Matrix is Empty!');

   if (High (X) <> High (Y)) then
      raise EMathError.Create ('Matrices must be the same Dimension to
Subtract!');

   N := High (X [0]);
   SetLength (Result, High (X) + 1, N + 1);
   for I := 0 to High (X) do
   begin
      if (High (X [I]) <> N) then
      begin
         Result := nil;
         raise EMathError.Create ('Matrices must be truly rectangular to
Subtract!');
      end;
      if (High (Y [I]) <> N) then
      begin
         Result := nil;
         raise EMathError.Create ('Matrices must be the same Dimension to
Subtract!');
      end;

      for J := 0 to N do
         Result [I, J] := X [I, J] - Y [I, J];
   end;
End;
```

SubVectors Function

Unit
ESBMaths2

Declaration
```
Function SubVectors(const X, Y: TDynFloatArray): TDynFloatArray;
```

Description
The Length of the resultant vector is that of the smaller of X and Y.

Implementation

```
function SubVectors (const X, Y: TDynFloatArray): TDynFloatArray;
var
   I: LongWord;
begin
   SetLength (Result, MinL (High (X), High (Y)) + 1);
   for I := 0 to High (Result) do
      Result [I] := X [I] - Y [I];
End;
```

TransposeMatrix Function

Unit
ESBMaths2

Declaration
```
Function TransposeMatrix(const X: TDynFloatMatrix): TDynFloatMatrix;
```

Description
Only works with Rectangular Matrices.

Implementation

```
function TransposeMatrix (const X: TDynFloatMatrix): TDynFloatMatrix;
overload;
var
   I, J: LongWord;
     XRows, XColumns: LongWord;
     XRectangular: Boolean;
begin
   Result := nil;
   MatrixDimensions (X, XRows, XColumns, XRectangular);
   if not XRectangular then
      raise EMathError.Create ('Matrix must be Rectangular');
   if (XRows = 0) or (XColumns = 0) then
      Exit;


   SetLength (Result, XColumns, XRows);
     for I := 0 to XRows - 1 do
        for J := 0 to XColumns - 1 do
         Result [J, I] := X [I, J];
End;
```

TDynFloatArray type

Unit
ESBMaths2

Declaration
```
TDynFloatArray = array of Extended;
```

TDynFloatMatrix type

Unit
ESBMaths2

Declaration
```
TDynFloatMatrix = array of TDynFloatArray;
```

TDynLIntArray type

Unit
ESBMaths2

Declaration
```
TDynLIntArray = array of LongInt;
```

TDynLIntMatrix type

Unit
ESBMaths2

Declaration
```
TDynLIntMatrix = array of TDynLIntArray;
```

TDynLWordArray type

Unit
ESBMaths2

Declaration
```
TDynLWordArray = array of LongWord;
```

TDynLWordMatrix type

Unit
ESBMaths2

Declaration
```
TDynLWordMatrix = array of TDynLWordArray;
```

Welcome to MyLib

...Write some nice words here...

Introduction

...write text here...