Dr. A. Geißler
geisslea@web.de

adreieck - area of triangles
akfrader - Autocorrelation estimation by the method of Rader
akima - Akima Interpolation
bezier - Cubic Bezier Spline (Interpolation)
biorythm - Calculation of Biorythm
bmpcolor - Reads RGB Windows 3.1 Bitmaps
bmpread - reads an uncompressed Windows 3.1 Bitmap with 8-Bit quantization depth
burg - AR-Model by the method of Burg
butter - BUTTERWORTH FILTER DESIGN IN Z-DOMAIN
calcdat - Addition of days to one or several dates
catman_read - Reads catman binary measurement data files
cauchy - Cauchy integral regarding Residuals
circfit - circle fitting
corrcoeff - Correlation Coefficient calculation
countobj – count objects in a binary image.
cp42cut - Read MGCplus Data files partially, recorded with CP42
cp42mea - Reads MGCplus binary measurement files completely
date2day - Get weekday of a date
dday - Difference in days between several dates
dec2bin - conversion of decimal numbers into binary representation
delaunay - Delaunay Triangulation
distance - Minimum distance between to lines
dither - Dithering of a GrayScale matrix
erfcomp - error function with complex input arguments
erfinv - Inverse Error Function
factor - Getting prime factors
feiertag - calculation of celebration days
findgrup - from the binary vector A the starting- and endpositions of groups with 1 are determined
findnext - Finding neighbours ...
fresnel - calculation of the fresnel integrals
gaussfit - optimal gauss curve through given data set
geo2gkk - conversion geograph. coordinates -> UTM-coord.
geo2gkn - conversion of geographical coord. into GK-N coordinates
gkk2geo - UTM-Coordinates -> geograph. Coord
gkn2geo - national GK coordinates -> geograph. coordinates
grad - gradient of function F
hitmiss - Skeleton of binary images
inerfc - Repeated Integrals of the error function
int2str - Convert integer to string
intriang - Are points within a triangle ?
ipolygon - Points within a closed polygon ?
isprime - checks number if they are primes
konfo - conformal linear transformation
leibnitz - Leibnitz-Sector formula (closed curves area)
mamufo - markov random function 1st order
markomat - Creation of a Markov Transition Matrix of 1st order
markov - Autocorrelation of a stationary Markov process
meacomp - Reading MGCplus-MEA files in a compressed form
mtlb_filter - One-dimensional digital filter
mtlb_freqz - Z-transform digital filter frequency response
multireg - Linear Regression for linear combined functions
nextdrei - Neighbour analysis of triangles and voronoy diagram
nullpass - counting and estimation of Zero Passes
ostern - calculation of the eastern day
plotmea - Plotting of MGCplus-MEA files
polyfit - Calculates the polynomial fitting
polyval - polynomial evaluation
ponbowl - Points on Bowl
ponci - Points on Circle
primes - Generate list of prime numbers
pwmratio - Pulswidth demodulation
randvek - Creation of a vector with length N with numbers from 1 ... N on random positions

adreieck - area of triangles

Calling Sequence

```
A = adreieck(plist)
```

Parameters

- **`plist`** : matrix with three rows. plist contains complex values, representing corners of triangles.
- **`A`** : vector, contains area of triangles

Description

PLIST is a matrix with 3 rows. Each column contains the corners of a triangle. A is a vector, which contains the areas of the triangles.

The function is quite easy. The main lines are: v1=plist(2,:) - plist(1,:); v2=plist(3,:) - plist(1,:); A=abs(0.5 .*imag(v1 .*conj(v2)));

$$A=\left|\frac{1}{2}imag(\vec{v}_1\cdot\underline{\vec{v}_2})\right| \text{ with } \vec{v}_2=\vec{p}_2-\vec{p}_1;\ \vec{v}_2=\vec{p}_3-\vec{p}_1$$

Examples

```
// Two triangles
pl=[0,1,(1+%i);0,3*%i,(4+3*%i)].'
A=adreieck(pl)
// Result should be ! 0.5   6 !
```

See Also

intriang, ipolygon

akfrader - Autocorrelation estimation by the method of Rader

Calling Sequence

```
y = akfrader(x,M)
```

Parameters

- **x** : row vector representing a signal
- **M** : Length of computed autocorrelation function
- **y** : The estimated autocorrelation [$R_{xx}(0)$, $R_{xx}(1)$, ... , $R_{xx}(M-1)$]

Description

The method of Rader is based upon a clever way of FFT transformations to realize the convolutions.

Quite fast and precise !

Examples

```
x=rand(1,5000,'normal');
y=akfrader(x,64);
k=0:1:63;
plot2d2(k,y);
```

See Also

burg, welch

Bibliography

Kammeyer, Karl Dirk, "DIGITALE SIGNALVERARBEITUNG : Filterung und Spektralanalyse" Teubner-Verlag 1989, Teubner-Studienb?her: Elektrotechnik , ISBN 3-519-06122-8

akima - Akima Interpolation

Calling Sequence

```
[xi,yi] = akima(x,y,K)
```

Parameters

- **x** : vector with abscissa values of function (must be sorted in ascending order)
- **y** : vector with ordinate values. Must have the same size as x.
- **K** : number of additional points between to abscissa values
- **xi** : new interpolated abscissa values
- **yi** : new interpolated ordinate values

Description

The given point (x,y) are approximated by 3rd order polynomials. The 2nd derivative is not considered (like with spline interpolation). Therefore wiggles are surpressed as far as possible.

If the input sequence consist of N points, then the result of interpolation consists of (N-1)*K points.

Examples

```
x=[1,2.5,3.2,4.8,5.9,6.2,7.7,8.3,9.3];
y=[0,0,1,1,2,2,3,3,4];
[xi,yi]=akima(x,y,5);
plot2d(x,y,style=-3);
plot2d(xi,yi,style=2);
```

See Also

bezier

bezier - Cubic Bezier Spline (Interpolation)

Calling Sequence

```
zi = bezier(z,NPOI,modus)
```

Parameters

- **z** : vector with complex values, which contains the so called weighing points.
- **NPOI** : number of interpolating values. Default value is 40.
- **modus** : a string, which specifies, if curves are closed ('c' or 'close') or open ('o' or 'open'). The default is 'close'.
- **zi** : cubic bezier spline represented as a vector with complex elements

Description

If the weighing points are not given as complex vector, it is possible to specify them by clicking with the mouse in a graphics window. If N weighing points are given, the interpolation result has N*(NPOI-1) values when modus='closed'. If modus='open', then the interpolation result has (N-2)*(NPOI-1) values.

Examples

```
z=rand(1,10,'normal') + %i .*rand(1,10,'normal');
zi=bezier(z,10,'open');
plot2d(real(z),imag(z),style=-3);
plot2d(real(zi),imag(zi),style=2);
xclick();
xbasc();
zi=bezier();
```

See Also

akima

biorythm - Calculation of Biorythm

Calling Sequence

```
BIO = biorythm(month,year,birth)
```

Parameters

- **`month`** : month for biorythm calculation
- **`year`** : year for biorythm calculation
- **`birth`** : array with three elements, representing a birth date. Format of the vector birth is [DD MM YYYY]
- **`BIO`** : array with 3 rows. The 1st row contains the "Body" curve, the 2nd row the "Soul" curve and the 3rd the "Brain" curve. Each column represents a day of the month, where the biorythm has been calculated for.

Description

Sometimes it is good to know, how you feel on a special day in future (examing days, wedding days etc.). Then you should have a look onto you "BIO" curves.

May be, Scilab helps you for your decision finding process, so that you shift your wedding day or you give up studies, when you see, that you are not fit on your examing days :-)

Examples

```
bio=biorythm(3,2006,[28,12,1966]) //my biorythm in march 2006
```

See Also

dday, calcdat

Used Function

dday

bmpcolor - Reads RGB Windows 3.1 Bitmaps

Calling Sequence

```
[yr,yg,yb] = bmpcolor(datei)
```

Parameters

- **`datei`** : complete filename of the RGB bitmap
- **`yr`** : Matrix with Red Channel
- **`yg`** : Matrix with Green Channel
- **`yb`** : Matrix with Blue Channel

Description

This function reads a file representing a Windows 3.1 RGB bitmap. The range of values of the outputs arguments is [0...255]. The matrices y_r, y_g, y_b are of type double.

If only one rhs-argument is used for function call, then the matrix y_r contains the luminace information. The luminance is calculated by the formula

$$y = 0.299 \cdot y_r + 0.587 \cdot y_g + 0.114 \cdot y_b$$

Examples

```
[yr,yg,yb]=bmpcolor();
```

See Also

bmpread

bmpread - reads an uncompressed Windows 3.1 Bitmap with 8-Bit quantization depth

Calling Sequence

```
[y,map] = bmpread(datei)
```

Parameters

- **`datei`** : complete filename of the bitmap (optional)
- **`y`** : The picture matrix. The range of values is [1...256]. Each element of y represents a row of the colormap.
- **`map`** : The colormap of the 8-Bit Bitmap. The range of values is [0...1]. The matrix map consists of 3 columns. The 1st column represents the RED, the 2nd the GREEN and the 3rd the BLUE part of a color.

Description

The SIVP toolbox offers more possibilites to read images. This function is usefull, when you want to modify it. For example reading in only a few rows of a picture.

Examples

```
[Y,MAP]=bmpread(); //Select an 8-Bit bitmap
f=gcf();
f.color_map=MAP;
Matplot(Y);
```

See Also

bmpcolor

burg - AR-Model by the method of Burg

Calling Sequence

```
[A,GAIN] = burg(x,R)
```

Parameters

- **x** : stationary random process, represented by a vector
- **R** : number of iterations. Default value is R=8
- **A** : denominator coefficients
- **GAIN** : The gain, represents the nominator

Description

x specifies a stationary random process. With the function burg you calculate the characterisitc values of an AR filter. It is assumed, that a white noise signal, filtered with these characteristics, has nearly the same statistical properties as x.

Therefore, the transfer function specified by the AR filter represents nearly the spectrum of the random process x. With [H,W]=mtlb_freqz(gain,A,N); [N is for example 256] you get the transfer function in the Fourier area.

Examples

```
x=rand(1,4000,'normal'); // Generate white gaussian noise
hz=iir(8,'bp','cheb2',[0.1,0.2],[1E-4,1E-4]);
y=flts(x,hz); //bandpass filtering
[A,Gain]=burg(y,11); //Burg estimation
[H,W]=mtlb_freqz(Gain,A,512);
plot2d(W,abs(H));
```

See Also

mtlb_freqz

Bibliography

Kammeyer, Karl Dirk : "DIGITALE SIGNALVERARBEITUNG : Filterung und Spektralanalyse", Teubner-Verlag 1989, Teubner-Studienbücher: Elektrotechnik ISBN 3-519-06122-8

butter - BUTTERWORTH FILTER DESIGN IN Z-DOMAIN

Calling Sequence

```
[B,A] = butter(n,filtertype,filterfreq)
```

Parameters

- **`n`** : filter order
- **`filtertype`** : Type of filter, specified by a string
- **`filterfreq`** : scalar or vector with 2 elements
- **`B`** : nominator coefficients
- **`A`** : denominator coefficients

Description

This function calculates the filter coefficients for a butterworth filter in the z-domain, so that the output value of the filter is given by:

$$a_1 \cdot y_n = b_1 \cdot x_n + b_2 x_{(n-1)} + \ldots + b_{(n_b+1)} \cdot x_{(n-n_b)} - a_2 \cdot y_{(n-1)} - \ldots - a_{(n_a+1)} \cdot y_{(n-n_a)}$$

This function emulates the MatLab Butter function from the signal toolbox. For the realization the SciLab-functions "iir", "coeff", "denom" and "numer" are used. If the filtertype is 'bp' or 'sb', then you need a vector for the argument filterfreq.

Examples

```
[B,A]=butter(8,'lp');
x=rand(1,300,'normal');
y=mtlb_filter(B,A,x);
t=0:1:299;
plot2d(t,x,style=3);
plot2d(t,y,style=2);
```

See Also

mtlb_filter

calcdat - Addition of days to one or several dates

Calling Sequence

```
    Qnew = calcdat(Q,anzday)
```

Parameters

- **`Q`** : Q is the date matrix. Every row represents a date. The 1st column contains the days, the 2nd the months and the 3rd the years
- **`anzday`** : scalar value, represents the number of days which should be added to the dates in Q
- **`Qnew`** : result of adding days

Description

Aquite simple algorithm.

Examples

```
Q=[28,12,1966;25,11,1965;12,8,1999];
Qnew=calcdat(Q,15000);
disp(Q);
disp(Qnew);
```

See Also

dday

catman_read - Reads catman binary measurement data files

Calling Sequence

```
[a1,a2] = catman_read(%file)
```

Parameters

- **%file** : complete filename of the catman measurement file. This input argument is optional.
- **a1** : Content of the global section. This structured variable contains information about filename , fileid (Must be greater than 5010 !), dataoffset (there, the raw data is contained), comment (The channel comment itself) , ReservedSTR (the reserved strings, only for catman 5.0, almost empty), noofchan (Number of channels), mcl (max. channel length), redufact (reduction factor, almost 0).
- **a2** : This structured variable contains the channel header data and the measurement data itself. The fields are: Channelnumber (catman database channel), ChannelLength (no. of values in the DB channel), ChannelName (Name of the DB channel), Unit , comment , format (0=numeric, 1=String, 2=Binary Object), datawidth (numeric = 8, string >= 8), datumzeit (Date/Time of measurement), header (Traceability data, must be decoded), linmode (Linearization Mode), userscale , DBSensorInfo , data (The REAL content of the database)

Description

CATMAN is a software family for DAQ purposes. It is optimized for DAQ with the amplifier systems MGCplus, MGCsplit and Spider8 of the company HBM GmbH (Hottinger Baldwin Messtechnik GmbH). Especially the software catman easy enables the fast and easy measurement with HBM hardware like Spider8 and MGCplus. Basis is TEDS based upon IEEE1451.4. Now you can direct import catman datafiles into SciLab.

ONLY CATMAN 4.5 and CATMAN 5.0 is supported !!! The catman online data format is not supported.

Examples

```
[a1,a2]=catman_read(); // Select a catman binary file and reads it
disp(a2);
disp(a2.ChannelName);
plot2d(a2(4).data);  // Plot the data of the 4th catman channel
```

See Also

cp42mea, cp42cut

cauchy - Cauchy integral regarding Residuals

Calling Sequence

```
W = cauchy(z,p,modus)
```

Parameters

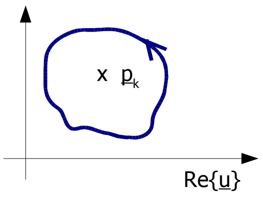
- **z** : z is a complex vector, describing a closed curve.
- **p** : p is a complex scalar, a complex vector or a complex matrix describing points (which are representing residuals)
- **modus** : The optional input Argument modus, default='y', is responsible for a nice graphical display
- **W** : W is the complex value of the integral and has the same size as p.

Description

We are calculating the ring integral of du/(u-p) along the curve specified in vector z. The function cauchy requires the function meshgrid in my library. "meshgrid" could be replaced by the transpose of the function resuld "ndgrid".

This function is useful, when you have to decide, whether a point p is inside or outside a closed curve z. If the point p is outside, then the cauchy integral is zero, otherwise it differs from zero (depending on the direction and the number of turnarounds.

$$W_k = \oint_{C(\underline{z})} \frac{d\underline{u}}{\underline{u} - \underline{p}_k}$$



Examples

```
zcurve=bezier(); //Draw a closed curve
x=0.1:0.1:0.9; y=x;
[X,Y]=meshgrid(x,y);
PZ=X + %i .*Y;
W=cauchy(zcurve,PZ);
```

See Also

meshgrid

Used Function

meshgrid

circfit - circle fitting

Calling Sequence

```
[xc,yc,R] = circfit(x,y)
```

Parameters

- **x** : abscissa value of some points
- **y** : ordinate values of some points, must have the same size as x
- **xc** : abscissa value of the center of the optimal circle
- **yc** : abscissa value of the center of the optimal circle
- **R** : radius of the optimal circle

Description

This function calculates the optimal circle in a set of distributed points, so that the squared error of deviations is minimal.

See Also

gaussfit, multireg

corrcoeff - Correlation Coefficient calculation

Calling Sequence

```
R = corrcoeff(x,y)
```

Parameters

- **x** : When two rhs arguments are used (x and y), x is a vector and has the same length as y. If only one rhs argument is used (no y), then x must be a matrix with at least two columns.
- **y** : a vector with the same length as x
- **R** : the correlation coefficient. If two rhs arguments are used, R is a scalar. If two rhs arguments are used, R is a square matrix with so many columns as the input matrix x. the element r(i,j) equals the correlation coefficient between signal i and signal j (columns i and j of input matrix x).

Description

The calculation of the correlation coefficients is done in the time area. Especially, when only one rhs is used, the strength of SciLab, MatLab or Octave can be seen:

In the sci-function, the correlation coefficient is calculated by:

```
zm1=x - meshgrid(mean(x,'r'),1:1:N);
W0=(zm1.')*zm1;
W1=sqrt(diag(W0)*diag(W0).');
R=W0 ./W1;
```

Examples

```
X=rand(1000,100,'normal');
R=corrcoeff(X);
k=0:1:99;
plot2d(k,diag(R));
xclick();
xbasc();
plot2d(k,diag(R,1));
```

See Also

akfrader, meshgrid

Used Function

meshgrid

countobj - count objects in a binary image

Calling Sequence

```
[objzahl,Kp_rest,cs,meanpt] = countobj(w)
```

Parameters

- **w** : matrix, representing a binary image with elements of {0,1}
- **objzahl** : number of counted objects
- **Kp_rest** : matrix with index information of contour data. It contains in the 1st row the start positions and in the 2nd row the number of values followed, which have been used for object counting.
- **cs** : the contour data of the binary image. This contour data was generated by the SciLab function contour2di.
- **meanpt** : center points of the objects, represented by a matrix with 2 rows and so many columns as objects. The 1st row contains the x and the 2nd row the y values of the object center point.

Description

The matrix w represents abinary image. It contains elements from the set {0,1}. This functions counts the objects represented by values "1". The tricky things with objects is always: How do you define an object ? This function takes always the enclosing shape. When you have a small ring inside a large ring, then the inner ring is not counted ! This function is based upon a "line integration" method, not on binary erosion or dilations !. From a binary image we determine the contour lines. Now we take random points from these curves and we check, if they are enclosed by other curves. If not, then the counting result is increased by on, otherwise not.

However, at the end of the function a graph is drawn showing the counting results.

Examples

```
w=bmpread();        // Read Bitmap (8 Bit)
wbin=bool2s(w>128); // create binary image
[ct,kprest,csdata, centers]=countobj(wbin);
// End of Demo
```

See Also

cauchy

Used Function

contour2di

cp42cut - Read MGCplus Data files partially, recorded with CP42

Calling Sequence

```
[filinf,CHINFO] = cp42cut(meafile,idx)
```

Parameters

- **`meafile`** : complete filename of the MGCplus measurement file
- **`idx`** : Row vector with two elements, containing the start and end measurement sample index
- **`filinf`** : a struct, containing the FileID, the number of channels "ChannelC", Byte/Line, the data format, the data offset, the time format, the file comment, the comment of the current parameterset, the information about the creation date and the number of used slots.
- **`CHINFO`** : a struct, containing information about the channel number, the scaling, offset, unit, signalling bit, subchannel, data, timeaxis, channel name and the channel code. Also the CHANNEL STATUS, including MGCplus LIMIT VALUES Switch State, CHANNEL ERROR, OVERFLOW are contained in the field status for each measurement channel . `Bit 0=Status Limit Value 1 , Bit 1=Status Limit Value 2 , Bit 2=Status Limit Value 3, Bit 3=Status Limit Value 4 , Bit 4=Overflow Gross Value, Bit 5=Overflow Net Value, Bit 6=calibration error , Bit 7=Change-Flag (amplifier scaling has changed)`

Description

Only CP42 measurement files are supported. These DAQ files may contain time information in following formats:
1) No Time Information
2) MGCplus Time (64 Bit)
3) NTP Time Format .
The Time Formats "USB Frame Count" and 32 Bit MGC device Time are not supported.

IMPORTANT: Only Measurement files with format 4 BYTE INT (Intel) are supported.

Examples

```
[a1,a2]=meacomp(2000); // Reads a measurement file as activity overview
plotmea(a1,a2); // Shows the measurement data
[cl1,cl2,cl3]=xclick(); // Select start index
[cr1,cr2,cr3]=xcklick(); // Select end index
[b1,b2]=cp42cut(a1.filename,[cl2,cr2]);
xbasc();
plotmea(b1,b2);
```

See Also

plotmea, cp42mea

cp42mea - Reads MGCplus binary measurement files completely

Calling Sequence

```
[filinf,CHINFO] = cp42mea(meafile)
```

Parameters

- **`meafile`** : full name of the MGCplus measurement file. Standard extension of the files is *.me*
- **`filinf`** : fileinfo is a struct, containing the FileID, the number of channels "ChannelC", Byte/Line, the data format, the data offset, the time format, the file comment, the comment of the current parameterset, the information about the creation date and the number of used slots.
- **`CHINFO`** : CHINFO is a struct, containing information about the channel number, the scaling, offset, unit, signalling bit, subchannel, data, timeaxis, channel name and the channel code. Also the CHANNEL STATUS, including MGCplus LIMIT VALUES Switch State, CHANNEL ERROR, OVERFLOW are contained in the field status for each measurement channel.`Bit 0: Status Limit Value 1; Bit 1: Status Limit Value 2; Bit 2: Status Limit Value 3; Bit 3: Status Limit Value 4 ; Bit 4: Overflow Gross Value; Bit 5: Overflow Net Value; Bit 6: calibration error ; Bit 7: Change-Flag (amplifier scaling has changed)`

Description

There is a lot of information in the mea files about channels and slots. Not all of these information is evaluated. Only CP42 measurement files are supported. These DAQ files may contain time information in following formats:
1) No Time Information
2) MGCplus Time (64 Bit)
3) NTP Time Format .
The Time Formats "USB Frame Count" and 32 Bit MGC device Time are not supported.

IMPORTANT: Only Measurement files with format 4 BYTE INT (Intel) are supported.

Examples

```
[a1,a2]=cp42mea(); // Opens a file open dialogue and starts reading
plotmea(a1,a2);
```

See Also

plotmea, cp42cut

date2day - Get weekday of a date

Calling Sequence

```
day = date2day(sdat)
```

Parameters

- **`sdat`** : row vector with at least 3 elements. The 1st three elements are used. They are: [Year, Month, Day, ...]
- **`day`** : a string containing the weekday

Description

The function checks, if the date is valid. Switching Years are considered.

Examples

```
date2day([2005,10,29])
```

See Also

calcdat, dday

dday - Difference in days between several dates

Calling Sequence

```
jd = dday(Q)
```

Parameters

- **Q** : The matrix Q contains the dates. Each row represents one date. In the 1st column is the day, in the 2nd column the month and in the 3rd the year
- **jd** : Difference in days. jd is a column vector. If the date matrix has N rows, jd has N-1 rows.

Description

A very usefull tool, for example the calcualtion of celebration days, bioryhms etc. You can check the results by use of the function calcdat.

Examples

```
Q=[25,11,1965;28,12,1966;26,12,1967];
jdn=dday(Q);
disp(jdn);
```

See Also

calcdat, biorythm

dec2bin - conversion of decimal numbers into binary representation

Calling Sequence

```
y = dec2bin(x)
```

Parameters

- **x** : a row vector decimal numbers.
- **y** : a matrix, representing the decimal numbers of x as binary numbers. Each column represents one decimal number. The 1st row of y contains the most signigicant bits, the last row the least significant bit.

Description

The elements of the input row vector are made positive and rounded. Only natural positive numbers are used for that function.

Examples

```
x=ceil(256 .*rand(1,10));
y=dec2bin(x);
disp(x);
disp(y);
```

delaunay - Delaunay Triangulation

Calling Sequence

```
[MP,R,v,tri] = delaunay(u)
tri = delaunay(u)
```

Parameters

- **u** : points specified by complex vector
- **MP** :Delaunay points, given by complex vector
- **R** : hullcircle-Radius of Delaunay points
- **v** : Matrix with points belonging to a triangle. The command "PLOT2D(REAL(ZLIST),IMAG(ZLIST))" draws the result of triangulation
- **tri** : Matrix with Indices of the points contained in Z. Each row describes a triangle

Description

when the function is called only with one output argument, the delaunay triangulation is shown in a graphical window.

This function is more or less for learning tasks. A better way is using mesh2d from the great "MetaNet"-Toolbox. This function has problems with regular distributed points, it works fine with points of random distribution.

Examples

```
p=rand(1,20,'normal') + %i .*rand(1,20,'normal');
tri=nextdrei(p);
```

See Also

nextdrei, ponci

Used Function

ponci

distance - Minimum distance between to lines

Calling Sequence

```
[p1,p2,entf] = distance(a,b,c,d)
```

Parameters

- **a** : Local Vector Line 1
- **b** : Direction Vector Line 1
- **c** : Local Vector Line 2
- **d** : Direction Vector Line 2
- **p1** : point on line 1 with minimum distance to line 2
- **p2** : point on line 2 with minimum distance to line 1
- **entf** : minimum distance between line 1 and 2

Description

This function makes quite simple operations:

$$\vec{r}=\begin{pmatrix}\vec{b}^T\cdot\vec{b} & -\vec{b}^T\cdot\vec{d}\\ \vec{b}^T\cdot\vec{d} & \vec{d}^T\cdot\vec{d}\end{pmatrix}^{-1}\cdot\begin{pmatrix}\vec{b}^T\cdot(\vec{c}-\vec{a})\\ \vec{d}^T\cdot(\vec{c}-\vec{a})\end{pmatrix}; \qquad \vec{p}_1=\vec{a}+r_1\cdot\vec{b};\ \vec{p}_2=\vec{c}+r_2\cdot\vec{d}$$

Examples

```
a=[1,1,0]; b=[0.5,2,1];
c=[0,1,2]; d=[2,0.5,3];
[p1,p2,entf]=distance(a,b,c,d);
disp(p1);
disp(p2);
disp(entf);
```

See Also

ponci, ponbowl

dither - Dithering of a GrayScale matrix

Calling Sequence

```
y = dither(x,method)
```

Parameters

- **x** : Matrix with elements with range of values [1...256], representing e. g. a grayscale image.
- **method** : string with 2 characters, determing the method of dithering
- **y** : dithered binary picture, matrix with elements with range of value {0,1}.

Description

Several mehtods of dithering are supported. Ordered Dithering methos are Bayer-1 (method='b1'), Bayer-2 (method='b2'), Bayer-3 (method='b3'), Stucki-1 (method='s1'), Stucki-2 (method='s2'). There is an additional ordered dithering method, which is based upon a randon matrix (method='rd').

When the 2nd rhs argument is choosen as method='rand', then random dithering is done (Default).

Examples

```
k=1:1:256;
[X,Y]=ndgrid(k,k);
Z=ceil(0.5 .*(X+Y)); // Generate a picture
f=gcf();
f.color_map=graycolormap(256);
Matplot(Z);
xclick();
xbasc();
zd=dither(Z,'s1'); // Do the dithering
f.color_map=graycolormap(2);
Matplot(full(zd)+1); // Show the dithered picture
```

See Also

bmpread, bmpcolor

erfcomp - error function with complex input arguments

Calling Sequence

```
E = erfcomp(z)
```

Parameters

- **z** : complex scalar, vector or matrix.
- **E** : The errorfunction for complex arguments. E has got the same size like z.

Description

The complex error function is usefull for defining other functions, like e. g. the fresnel function.

Examples

```
z=[1-%i,1,1+%i,%i,-1+%i,-1,-1-%i,-%i];
EC=erfcomp(z);
disp(EC);
```

See Also

fresnel, erfinv

Bibliography

Abramovitz / Stegun: "Handbook of mathematical functions"

erfinv - Inverse Error Function

Calling Sequence

```
x = erfinv(y)
```

Parameters

- **y** : real scalar, vector or matrix. Range of values is [-1...+1]
- **x** : result of the inverse error function. x has same size as x.

Description

X = ERFINV(Y) is the inverse error function for each element of Y. The inverse error function satisfies y = erf(x), for $-1 \le y \le 1$ and $-\infty \le x \le +\infty$. This function has been converted from MatLab.

Examples

```
u=erfinv(0.75)
erf(u)
```

See Also

erfcomp, fresnel

Bibliography

Abramovitz / Stegun: "Handbook of mathematical functions"

factor - Getting prime factors

Calling Sequence

```
f = factor(n)
```

Parameters

- **n** : positive natural scalar, range of values [0...2^{32}]
- **f** : row vector, containing the prime factors of n

Description

This function uses the simple sieve approach. It may require large memory allocation if the number given is too big. Technically it is possible to improve this algorithm, allocating less memory for most cases and resulting in a faster execution time. However, it will still have problems in the worst case, so we choose to impose an upper bound on the input number and error out for $n > 2^{32}$. This function has been imported from MatLab.

Examples

```
pfc=factor(102);
disp(pfc);
```

See Also

isprime

feiertag - calculation of celebration days

Calling Sequence

```
[FEIERTAGNAME,Qfeier] = feiertag(j,bdg,kath)
```

Parameters

- **j** : The year, for which celebration days are calculated
- **bdg** : string determing the german countries. Following abbreviations are valid.
  ```
  'BAY' = Bayern, 'BRB' = Brandenburg, 'BAW' = Baden-Württemberg,
  'HES' = Hessen (Default), 'MVP' = Mecklenburg-Vorpommern,
  'NRW' = Nordrhein-Westfalen, 'RPF' = Rheinland-Pfalz, 'SAC' = Sachsen,
  'SAA' = Sachsen-Anhalt, 'SAR' = Saarland, 'THÜ' = Thüringen.
  ```
- **kath** : 0 for non-catholic regions, 1 for catholic regions.
- **FEIERTAGNAME** : string-array with names of celebration days (in german, sorry)
- **Qfeier** : matrix with three columns [day, month, year]. The rows of this matrix and the string array FEIERTAGNAME are belonging together.

Description

This function is very powerful. It shows very clearly, how the celebration days are calculated. Many celebration days are dependent on eastern, others are fixed dates.

The function has been converted from an old excel macro from the time, when people had used Windows 3.1 [may be, it was before your times ;-)]

Examples

```
[ft1,dt1]=feiertag(2006,'HES',0);
disp(ft1);
disp(ft2);
```

See Also

ostern, date2day

findgrup - from the binary vector A the starting- and endpositions of groups with 1 are determined

Calling Sequence

```
K = findgrup(a,modus)
```

Parameters

- **a** : row vector containing elements with range of value {0,1}
- **modus** : scalar value, range of value {0,1}. if modus is 0 (default), leading and trailing 1-groups are not considered.
- **K** : matrix with two rows. The 1st row contains the start index of the 1-group, the 2nd row the last index of a 1-group.

Description

This is a very usefull tool for e. g. selecting areas of interest of time functions.

The function is converted from my MatLab functions

Examples

```
a=[1,1,1,0,0,0,1,1,0,0,0,1,1,1];
ki1=findgrup(a);
disp(ki1);
ki2=findgrup(a,1); //with Leading and trailing groups
disp(ki2);
```

See Also

findnext, pwmratio

findnext - Finding neighbours ...

Calling Sequence

```
bound = findnext(v1,v2)
```

Parameters

- **v1** : vector with real numbers
- **v2** : vector with real numbers, elements must be sorted in ascending order.
- **bound** : matrix with so many rows as length of v2. The 1st column contains the next smallest and the 2nd column the next larger elements of v1 regarding v2.

Description

The vector v2 contains numbers and the vector v1 is scanned for the next smallest and the next larger value. These both values are written to the matrix BOUNDS (2 columns). The 1st column contains the next smallest, the 2nd column the next larger values.

If there is no next smallest value, the the lower boundary is the minimal value of v1. Also if there is no next larger value, then the maximum of v1 is used as result.

Examples

```
v1=rand(1,10);
v2=rand(1,5);
v2=gsort(v2);
v2=v2($:-1:1);
bds=findnext(v1,v2);
disp(v2);
disp(bds);
```

See Also

findgrup

fresnel - calculation of the fresnel integrals

Calling Sequence

```
    K = fresnel(x)
```

Parameters

- **x** : scalar, vector or matrix with real values
- **K** : the fresnel integral $K(x)=C(x)+i\cdot S(x)$

Description

The calculation is not done by integration but by evaluation of approximation and series.

Examples

```
x=linspace(-10,10,2000);
K=fresnel(x);
plot2d(real(K),imag(K),style=2);
```

See Also

erfcomp

Bibliography

Abramovitz / Stegun: "Handbook of mathematical functions"

gaussfit - optimal gauss curve through given data set

Calling Sequence

```
[Q,CC] = gaussfit(x,y)
```

Parameters

- **x** : abscissa values of points (vector or matrix)
- **y** : ordinate values of points (same size as x)
- **Q** : vector with gauss curve parameters, representing the optimal gauss curve
- **CC** : approximation qualitiy. $0 \le CC \le 1$

Description

The data set $\{x_k, y_k\}$ should be approximated by an optimal Gauss curve of the form $y = k_0 \cdot \exp(-(x-\mu)^2 / (2 \cdot \sigma^2))$ .The quadratic deviations are optimized. Base of optimization is the equation $e^{(b_2 \cdot x^2 + b_1 \cdot x + b_0)}$.

CC represents the approximation quality $0 \le CC \le 1$, the matrix Q contains in the 1st row the coefficients b_2, b_1, b_0 and in the 2nd row the parameters. k_0, μ, σ of the optimal gauss curve. Only data points with y>0 are used for approximation !

Examples

```
x=rand(1,40,'normal');
mueh=1.2; sgma=1.8;
y=exp( -((x-mueh) .^2) ./(2 .*(sgma^2))) ./sqrt(2 .*(sgma^2));
[Q,CC]=gaussfit(x,y);
```

See Also

multireg

geo2gkk - conversion geograph. coordinates → UTM-coord.

Calling Sequence

```
z = geo2gkk(w2,w1)
```

Parameters

- **w2** : Vector with longitude (DEG)
- **w1** : Vector with lattitude (DEG
- **z** : complex Vector with Gauss-Krüger coord. Real part=right value, Imag. part=height value.

Description

This function uses the constants of the earth ellipsoide, how it is used by the "Deutsches Landesvermessungsamt". In the SCI file, you can see also the constants used by WGS84.

The strength of this function is, that it also delivers very good results for points with large lattitude (north and south pole regions) and that it can be used also for large values of longitudes. Very complex mathematical operations are performed.

Examples

```
lo1=8;
la1=49;
Z=geo2gkk(lo1,la1);
[lo2,la2]=gkk2geo(Z);
disp(lo2-lo1);
disp(la2-la1);
```

See Also

gkk2geo, geo2gkn

geo2gkn - conversion of geographical coord. into GK-N coordinates

Calling Sequence

```
[z,NRout] = geo2gkn(w2,w1,NR)
```

Parameters

- **`w2`** : Vector with longitude values (DEG)
- **`w1`** : Vector with lattitude values (DEG)
- **`NR`** : meridional stripe number. Argument is optional. If it is not used, it is calculated in relation to the longitude value. In Germany, meridional stripe numbers 3, 4 and 5 are used.
- **`z`** : complex vector with Gauss-Krüger coordinates. Real part = Right Value, Imag. part = height value.
- **`NRout`** : meridional stripe number for the calculated Gauss-Krüger value.

Description

The function geo2gkn calculates Gauss-Krüger Coordinates in the cartesian Potsdam-Rauenberg-DHDN format. By changing the values of the constants of the earth ellipsoide, other cartesian formats can achieved.

However: A transformation is not done, because it is just funny. The background behind all coordinate transformations are further mathematical operations, like calculating distances, directions etc.

Examples

```
[Z1,NR]=geo2gkn(8,49,3);
[Z2,NR]=geo2gkn(10,51,3);
d=abs(Z2-Z1)/1000; // Distance between points in km
disp(d);
```

See Also

gkn2geo, tkblatt

Used Function

geo2gkk

gkk2geo - UTM-Coordinates → geograph. Coord

Calling Sequence

```
    [lam,phi] = gkk2geo(Z)
```

Parameters

- **`Z`** : complex Vector with Gauss-Krüger-coord. Real part=Right value, Imag. part=Height value.
- **`lam`** : Vector with longitudes (DEG)
- **`phi`** : Vector with lattitude (DEG)

Description

gkk2geo is the inverse operation of geo2gkk. A very dedicated algorithm is used, therefore it is precise independent on the positions used.

Examples

```
lo1=8;
la1=49;
Z=geo2gkk(lo1,la1);
[lo2,la2]=gkk2geo(Z);
disp(lo2-lo1);
disp(la2-la1);
```

See Also

geo2gkk

gkn2geo - national GK coordinates → geograph. coordinates

Calling Sequence

```
[w1,w2] = gkn2geo(z,NR)
```

Parameters

- **z** : complex vector with Gauss-Krüger coordinates. Real part=Right value, Imag. part =height value
- **NR** : number of meridional stripe
- **w1** : Vector with longitude values (DEG)
- **w2** : Vector with lattitude values (DEG)

Description

national Gauss-Krüger coordinates can be converted back to geographical coordinates. This function is the inverse of geo2gkn.

Examples

```
w1a=8; w2a=49;
[Z,NR]=geo2gkn(w1a,w2a);
[w1b,w2b]=gkn2geo(Z,NR);
disp(w1b-w1a);
disp(w2b-w2a);
```

See Also

geo2gkn, gkk2geo

Used Function

gkk2geo

grad - gradient of function F

Calling Sequence

```
[fx,fy] = grad(F,dx,dy)
```

Parameters

- **F** : matrix, representing a function F(x,y)
- **dx** : optional parameter, default is 1. dx can be a scalar, then it represents the distance between equally spaced abscissa values. If it is a vector, then it represents the abscissa values x itself.
- **dy** : optional. If it is not used, then dx is the same as dx. dx must be in that case a scalar (distance between equally spaced abscissa value), otherwise an error is reported.
- **fx** : the derivative in x direction
- **fy** : the derivative in y direction

Description

grad works with 2-dimensional arrays. higher dimensions are not supported.

Examples

```
x=cumsum(rand(1,30));
y=cumsum(rand(1,40));
[X,Y]=meshgrid(x,y);
Z=sin(0.1 .*X - 0.2 .*Y);
[fx,fy]=grad(Z,x,y);
```

See Also

meshgrid

hitmiss - Skeleton of binary images

Calling Sequence

```
[Qout,WLE,WC,WS] = hitmiss(Qin)
```

Parameters

- **`Qin`** : Binary Matrix with elements of {0,1}.
- **`Qout`** : The skeleton of the input image.
- **`WLE`** : matrix with line end points
- **`WC`** : matrix with crossing points
- **`WS`** : matrix with singular points

Description

This function implements a hit and miss algorithm which can be used to create skeletons of a binary input image.

Examples

```
x=linspace(-%pi,%pi,200); y=x;
[X,Y]=meshgrid(x,y);
R=sqrt(X .^2 + Y .^2);
Z=sin(4 .*(R .^2));
Zbin=bool2s(Z>0.75);
[Zout,w1,w2,w3]=hitmiss(Zbin);
```

See Also

maskand, shiftmat

Used Function

maskand, shiftmat

inerfc - Repeated Integrals of the error function

Calling Sequence

```
w = inerfc(z,n)
```

Parameters

- **z** : complex vector or complex matrix
- **n** : order of the repeated integral
- **w** : result of the repeated error function. Same size as z.

Description

This function is partially used for field propagation estimation. Very special, not useful for everyone.

The function works recursive.

Examples

```
z=rand(2,5,'normal') + %i .*rand(2,5,'normal');
w=inerfc(z,4);
disp(w);
```

See Also

erfcomp

Bibliography

Abramovitz / Stegun: "Handbook of mathematical functions"

Used Function

erfcomp

int2str - Convert integer to string

Calling Sequence

```
s = int2str(x)
```

Parameters

- **x** : a matrix with real values
- **s** : string matrix

Description

S = INT2STR(X) rounds the elements of the matrix X to integers and converts the result into a string matrix. NaN and Inf values are represented by the value -2147483648

Examples

```
x=40 .*rand(3,4);
s=int2str(x);
disp(s);
```

intriang - Are points within a triangle ?

Calling Sequence

```
Q = intriang(ptri,points)
```

Parameters

- **ptri** : complex vector of length 3, contains corner of a triangle.
- **points** : a complex vector or a matrix with points
- **Q** : binary Matrix with the same dimension like the matrix or vector POINTS. A 1 is used for points, which are within or on the corner of the triangle.

Description

PTRI is a complex vector of length 3, which contains the corners of a triangle. POINTS is a complex vector or a matrix with points, which are checked, if they are within the triangle. Q is a binary Matrix with the same dimension like the matrix or vector POINTS. A 1 is used for points, which are within or on the corner of the triangle.

The decision is done by the checking the sum of areas, which are build from the point and the triangle.

Examples

```
pl=[0,4,(3+2*%i)] + (2+3*%i)
[X,Y]=ndgrid(0.3:0.5:7,0.3:0.5:7);
points=X + %i .*Y;
Q=intriang(pl,points);
// Draw the triangle
plot2d(real([pl,pl(1)]),imag([pl,pl(1)]),style=2);
X1=X(Q==1); Y1=Y(Q==1);
X2=X(Q==0); Y2=Y(Q==0);
// Plot the points
plot2d(X1,Y1,style=-3);
plot2d(X2,Y2,style=-4);
// end of example
```

See Also

adreieck

Used Function

adreieck

ipolygon - Points within a closed polygon ?

Calling Sequence

```
Q = ipolygon(p,points)
```

Parameters

- **p** : complex points specifying a curve
- **points** : complex points (vector or matrix)
- **Q** : matrix with the same size as points. Elements of Q are 0 or 1, depending on the position of points regarding the curve specified by p.

Description

From the complex points P the convex hull is calculated. (P can even specify the convex hull itself). The complex POINTS (Matrix or Vektor) are checked, whether they are located on the border or within the polygon. Result is a boolean matrix with the same dimension as POINT. ipolygon uses functions intriang (checks if a point is within a triangle) and adreieck (calculate area of triangle). The complete function makes at 1st a delaunay triangulation, then it is checked, whether the points are in one of these triangles.

Examples

```
x=0.1:0.1:0.9;
y=x;
[X,Y]=ndgrid(x,y);
Z=X + %i .*Y; // POINTS
phi=0:30:360;
p=(0.5 + %i .*0.5) + 0.3 .*exp(%i .*%pi .*phi/180);
Q=ipolygon(p,Z);
Q1=Z(Q==1);
Q2=Z(Q==0);
plot2d(real(Q2),imag(Q2),style=-3);
plot2d(real(Q1),imag(Q1),style=-4);
plot2d(real(p),imag(p),style=2);
```

See Also

delaunay

Used Function

delaunay, intriang

isprime - checks number if they are primes

Calling Sequence

```
I = isprime(a)
```

Parameters

- **a** : scalar, vector or matrix with positive natural numbers
- **I** : matrix with the same size as a. The elements are 0 or 1. If $a_{i,j}$ is prime, then $I_{i,j}$ is 1, otherwise 0.

Description

returns 1s for the prime elements in matrix P and 0s for the others.

This function was converted from a Matlab M File

Examples

```
a=round(30 .*rand(10,4));
P=isprime(a);
disp(a);
disp(P);
```

See Also

factor

konfo - conformal linear transformation

Calling Sequence

```
y = konfo(x,a)
```

Parameters

- **x** : matrix or vector with complex values
- **a** : matrix with 3 rows and 2 columns [3x2] with complex numbers.
- **y** : transformation of points specified by x by the transformation rule specified in matrix a.

Description

The matrix A (3x2) contains in the 1st column the z-values and in the 2nd column the w-values, on which the z values are transformed. On the complex values of x this tranformation is used

When you want to realize an inversion, then a is build up e.g. by
a=[1, 1; 1+%i, (1-%i)/2; 1-%i, (1+%i)/2]

Examples

```
a=[1,1;1+%i,(1-%i)/2;1-%i,(1+%i)/2];
x=(-10:0.1:10) + %i;
y=konfo(x,a);
```

leibnitz - Leibnitz-Sector formula (closed curves area)

Calling Sequence

```
A = leibnitz(x,y)
```

Parameters

- **x** : x-values of a closed curve.
- **y** : y-values of a closed curve. y is optional. If it is not used, the closed curve is represented by a complex vector x or by a contour-matrix with 2 rows.
- **A** : The area of the closed curve

Description

This function is used to calculate the area of a closed curve. Several calling conventions can be used:

1) A=LEIBNITZ(Z) ,
Z is a matrix with 2 rows. The 1st row equals x, the 2nd equals y.

2) A=LEIBNITZ(CVEK), CVEK is a complex Vector,
Real part equals x, imaginary part equals y.

Examples

```
phi=0:1:360;
z=exp(%i .*%pi .*phi ./180); // Circle with radius 1
A1=leibnitz(z);
disp(A1);
A2=leibnitz(real(z),imag(z));
disp(A2);
cs=[1,real(z);length(z),imag(z)];
A3=leibnitz(cs);
disp(A3);
```

See Also

simpson

mamufo - markov random function 1st order

Calling Sequence

```
y = mamufo(Q,xi,tnorm)
```

Parameters

- **Q** : probability matrix (transition prob.)
- **xi** : vector with possible values
- **tnorm** : time axis
- **y** : markov process as vector

Examples

```
S=markomat(5,0.8,'updo') // create transistion prop matrix
k=0:1:999;
xv=-2:1:2;
y=mamufo(S,xv,k);
plot2d(k,y);
```

See Also

markomat, markov

markomat - Creation of a Markov Transition Matrix of 1st order

Calling Sequence

```
Smark = markomat(N,pself,struc)
```

Parameters

- **`N`** : size of transition matrix
- **`pself`** : self propability
- **`struc`** : string. If it is 'updo', the transitions from element 1 to N ore reverse are disabled.
- **`Smark`** : the transition probability matrix

Description

This function creates a transition probabilty matrix for 1st order markov processes. The matrix can be used together with the function mamufo to generate a random process or it can be used in the function markov to determine the autocorrelation of markov processes with that transition probability.

Examples

```
S1=markomat(5,0.9,'updo');
S2=markomat(5,0.9,'both');
k=0:1:499;
y1=mamufo(S1,-2:1:2,k);
y2=mamufo(S2,-2:1:2,k);
plot2d(k,y1,style=2);
plot2d(k,y2,style=3);
```

See Also

mamufo, markov

markov - Autocorrelation of a stationary Markov process

Calling Sequence

```
[p,rxx] = markov(Q,merkmal,z)
```

Parameters

- **Q** : probability matrix
- **merkmal** : values of random process
- **z** : Time'-Difference for ACF
- **p** : probability of states
- **rxx** : the auto correlation function

Description

the autocorrelation is really computed, not estimated here !

$$\vec{p}=(\boldsymbol{Q}^T+\boldsymbol{U}-\boldsymbol{E})^{-1}\cdot\vec{I}$$

$$R_{xx}(k)=\vec{v}^T((\boldsymbol{Q}^T)^k\cdot(\vec{v}^T\cdot\vec{p})^T)$$

U is the unitary matrix (contains 1 on every position) with the same size as Q.

E is the identity matrix with the same size as **Q** and $\vec{I}$ is a unitary vector with the length equal to the column numbers of **Q**.

Examples

```
S=markomat(5,0.9,'updo')
v=-2:1:2;
z=0:1:63;
[P,rxx]=markov(S,v,z);
disp(P);
plot2d2(z,rxx);
// end of example markov
```

See Also

mamufo, markomat

meacomp - Reading MGCplus-MEA files in a compressed form

Calling Sequence

```
[fileinf,CHINFO] = meacomp(meafile,compfactor)
```

Parameters

- **`meafile`** : full pathname of the MGCplus measurement file
- **`compfactor`** : Compression factor. The bigger the factor, the faster is the reading of large meafiles. It contains the number of measurement samples per reading block.
- **`fileinf`** : structured variable with measurement file information
- **`CHINFO`** : structured variable with channel information. Also contained is a channel activity information (from 0% to 100%), so that you can see quite easy, where somethings happens.

Description

MEACOMP reads the values of the CP42 measurement file. A block containing all channels/signals is read. The number of samples is specified by the variable compression. From every channel/signal the max. and min. value are subtracted, so that you get an information about "signal activity" in each block.

The information about the file is stored in the struct variable FILEINFO, the information about channels and signals is contained in the variable CHINFO. CHINFO(k).data contains the "compressed" activity information, CHINFO(k).timeax contains the sample index.

Examples

```
[a1,a2]=meacomp(3000);    // Reading large meafile
plotmea(a1,a2);           // Plot the measurement data.
[b1,b2]=cp42cut(a1.filename,[70000,80000]);
xbasc();
plotmea(b1,b2);
```

See Also

plotmea, cp42cut

meshgrid - Generation of X and Y arrays for 3-D plots.

Calling Sequence

```
[xx,yy,zz] = meshgrid(x,y,z)
```

Parameters

- **x** : vector x direction
- **y** : y-vector
- **z** : z-vector
- **xx** : X-matrix
- **yy** : Y-matrix
- **zz** : Z-matrix

Description

[X,Y] = MESHGRID(x,y) transforms the domain specified by vectors x and y into arrays X and Y that can be used for the evaluation of functions of two variables and 3-D surface plots. The rows of the output array X are copies of the vector x and the columns of the output array Y are copies of the vector y.

[X,Y] = MESHGRID(x) is an abbreviation for [X,Y] = MESHGRID(x,x). [X,Y,Z] = MESHGRID(x,y,z) produces packed 3-D arrays that can be used to evaluate functions of three variables and 3-D volumetric plots

Examples

```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
```

mtlb_filter - One-dimensional digital filter

Calling Sequence

```
y = mtlb_filter(B,A,x)
```

Parameters

- **B** : nominator coefficients
- **A** : denominator coefficients
- **x** : input sequence (vector)
- **y** : filtered output sequence (vector)

Description

Y=MTLB_FILTER(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y. The filter is a "Direct Form II Transposed" implementation of the standard difference equation:

$$a_1 \cdot y_n = b_1 \cdot x_n + b_2 x_{(n-1)} + \ldots + b_{(n_b+1)} \cdot x_{(n-n_b)} - a_2 \cdot y_{(n-1)} - \ldots - a_{(n_a+1)} \cdot y_{(n-n_a)}$$

If a_1 is not equal to 1, FILTER normalizes the filter coefficients by a_1.

The function mtlb_filter can be replaced by the powerful SciLab function flts. mtlb_filter is just for the MatLab users starting first work with SciLab.

Examples

```
[B,A]=butter(8,0.1);
t=0:1:999;
x=rand(1,1000,'normal');
y=mtlb_filter(B,A,x);
plot2d(t,x,style=3);
plot2d(t,y,style=2);
```

See Also

butter, mtlb_freqz

mtlb_freqz - Z-transform digital filter frequency response

Calling Sequence

```
[h,w] = mtlb_freqz(b,a,n)
```

Parameters

- **b** : vector with nominator coefficients
- **a** : vector with denominator coefficients
- **n** : length of frequency response vector
- **h** : complex frequency response (length n)
- **w** : frequency vector (length n)

Description

[H,W]=mtlb_freqz(B,A,N) returns the N-point frequency vector W and the N-point complex frequency response vector G of the filter B/A: given numerator and denominator coefficients in vectors B and A. The frequency response is evaluated at N points equally spaced around the upper half of the unit circle.

This function has been imported from MatLab and one or two improvements for SciLab has been done.

Examples

```
[B,A]=butter(8,0.25);
[H,W]=freqz(B,A,256);
f=gcf();
subplot(211);
plot2d(W,abs(H),style=3);
subplot(212);
plot2d(W,atan(imag(H) ./real(H)),style=3);
disp(length(H));
```

See Also

mtlb_filter, butter

multireg - Linear Regression for linear combined functions

Calling Sequence

```
[a,f] = multireg(X,Y,q,adone)
```

Parameters

- **X** : Matrix. Each column represents a variable. The number of rows represents the amount of statistical samples. Each row represents a Tupel (x1,x2,...xm).
- **Y** : The function values belonging to the tupels (x1,x2,...,xm). Y can be a column vector or a matrix
- **q** : Selektion vector. Specifies the coefficients, which should be calculated.
- **adone** : Vector with predefined coefficients. The number of elements of ADONE + the number of elements of Q must be m.
- **a** : a is a Vector (or a Matrix, depends on Y) with coefficients. The solution vector a has m+1 coefficients.
- **f** : The vector f (or matrix) contains the error of the regression

Description

more or less a quite simple linear equation system task. The squared deviations are minimized. Nice feature for this function: The user can determine the coefficients he wants to optimize.

Several possibilities are supported: (1): Global determination of optimal parameters. The complete equation system is solved. (2): The optimal solution is calculated recursevely. The vector Q specifies the sequence.

Examples

```
x=rand(10,5);
a=[3;4;2;1;8];
y=x*a-3;
[aopt1,f1]=multireg(x,y)
//Now, only coeff. 1,2 and 4 are optimized.
[aopt2,f2]=multireg(x,y,[1,2,4])
// Coeffs 1, 2 and 4 are calculated.
// Coeffs 3 and 5 are set to 10 and 11
[aopt3,f3]=multireg(x,y,[1,2,4],[10,11])
```

See Also

gaussfit

nextdrei - Neighbour analysis of triangles and voronoy diagram

Calling Sequence

```
[Q,VS,zlist] = nextdrei(tri)
```

Parameters

- **tri** : Indexmatrix of Delaunay triangle points or complex Vector, contains points in R²
- **Q** : neighbour matrix. Each column of Q represents a triangle, which is specified by the indexmatrix TRI
- **VS** : Matrix with 2 rows. Each column represents a voronoy line.
- **zlist** : Matrix with 4 rows. The last row is identical to the 1st row. Each column describes a delaunay triangle

Description

If only the index matrix of delaunay points is given, then you get the neighbourmatrix of the delaunay triangles. If you use complex points as input argument, and you have less than two output arguments, then you will see also a graphical display of delaunay lines and voronoy diagram.

Triangulation works fine with irregualry spaced points. When you have installed the „metanet toolbox", then you do not really need this function.

Examples

```
p=rand(1,20,'normal') + %i .*rand(1,20,'normal');
[Q,vl,dl]=nextdrei(p);
plot2d(real(p),imag(p),style=-3);
// Now, Voronoy Lines will be drawn
plot2d(real(vl),imag(vl),style=2*ones(1,size(vl,2)));
// Now, Delaunay Triangles will be drawn
plot2d(real(dl),imag(dl),style=3*ones(1,size(dl,2)));
xclick();
xbasc();
TRI1=delaunay(p);
Q=nextdrei(TRI1);
```

See Also

delaunay

nullpass - counting and estimation of Zero Passes

Calling Sequence

```
[N2,N1] = nullpass(x,NKO)
```

Parameters

- **x** : random processes (vector or matrix). If x is a matrix, a random process is contained in a row.
- **NKO** : number of autocorrelation values
- **N2** : estimated number of zero passes
- **N1** : counted zero passes. Optional rhs argument, the default is 128.

Description

counting and estimation (based on autocorrelation) of the number of Zero passes of a random process. NKO is the number of values of the autocorrelation function in the positive plane. X my be a matrix or a vector. If X is a Matrix, each row vector is analyzed.

This function is for didactic purposes. It demonstrates the estimation of zero crossings of a stationary random process by evaluating the autocorrelation function. The estimated number of zero crossings is estimated by

$$\widetilde{N}=\frac{1}{\pi}\cdot\sqrt{R_{xx}''(0)/R_{xx}(0)}$$

Examples

```
// Create a low pass filtered stationary process
[B,A]=butter(8,0.08);
x=rand(1,4000,'normal');
y=mtlb_filter(B,A,x);
XY=[x;y];
[N2,N1]=nullpass(XY);
disp(N2); // estimated
disp(N1); // counted
```

See Also

akfrader, mtlb_filter

Bibliography

Papoulis, "Signal Analysis"

Used Function

akfrader

ostern - calculation of the eastern day

Calling Sequence

```
[Q,os] = ostern(j)
```

Parameters

- **j** : year. Years may be within the range 1583 - 2299
- **Q** : Date of eastern sunday in the form [day, month, year]
- **os** : difference between 1st of January and eastern sunday in days.

Description

The eastern date is important for the calculation of some other celebration days. This function is fundamental for the function "feiertag".

Examples

```
[Q,os]=ostern(2006)
```

See Also

feiertag, werktage

plotmea - Plotting of MGCplus-MEA files

Calling Sequence

```
plotmea(a1,a2,idx)
```

Parameters

- **`a1`** : contains the file information structure of the mea file.
- **`a2`** : contains the channel information including also measurement and traceability data
- **`idx`** : with this parameter you can determine, which channels are plotted. If you do not use this optional argument, all channels are plotted in one window !

Description

With the function cp42cut, cp42mea and meacomp it is possible to read MGCplus measurement files. With plotmea it is very easy to have a look on the measurement data.

MGCplus is a multifunctional multichannel data acquisition system, ideal to measure mechanical quantities.

Examples

```
[a1,a2]=cp42mea(); // Select a meafile to be read
plotmea(a1,a2);
```

See Also

cp42mea

polyfit - Calculates the polynomial fitting

Calling Sequence

```
[SSE,r,b] = polyfit(xx,yy,p)
```

Parameters

- **xx** : x-value of data sets
- **yy** : y-value of data sets
- **p** : polynomial degree
- **SSE** : squared sum error
- **r** : correlation coefficient
- **b** : optimal polynomial coefficients

Description

A classcial linear equation system to minimze the quadratic errors is solved.

Examples

```
x=rand(1,20,'normal');
a=[3,2,-4,1.5];
y=polyval(a,x);
// cubic polynomial
[SSE,r,b]=polyfit(x,y,3)
// square polynomial
[SSE,r,b]=polyfit(x,y,2)
```

See Also

polyval

Used Function

vandermonde

polyval - polynomial evaluation

Calling Sequence

```
y = polyval(C,x)
```

Parameters

- **C** : Polynomial coefficients (vector)
- **x** : Scalar, vector matrix with values, where the polynial should be evaluated
- **y** : Result of polynomial evaluation

Description

POLYVAL behaves like the matlab function polyval. But it is realized with standard SciLab functions "poly" and "horner". C (length=N+1) builds a polynomial with the form

$$y = \sum_{k=1}^{k=N+1} c_k \cdot x^{(N+1-k)}$$

Examples

```
a=[3,2,-4,1.5];
x=-10:1:10;
y=polyval(a,x)
```

See Also

polyfit, multireg

ponbowl - Points on Bowl

Calling Sequence

```
[mp,r] = ponbowl(puks)
```

Parameters

- **puks** : PUKS=[x1 y1 z1;x2 y2 z2;...;x4 y4 z4] specifies four points (on a bowl)
- **mp** : bowl center
- **r** : bowl radius

Description

4 points define a the surface of a bowl. These points are given in a matrix:
PUKS=[x1 y1 z1;x2 y2 z2;...;x4 y4 z4] .
MP is the center of the bowl, R its radius.
If the 4 points are laying on a circle, then the linear equation system is not defined anymore.

Examples

```
p=[1,1,1;3,8,10;-4,-5,-8;-3,6,-2]
[mp,r]=ponbowl(p)
// Quite simple example
```

See Also

ponci, distance

ponci - Points on Circle

Calling Sequence

```
[mp,r] = ponci(puks)
```

Parameters

- **`puks`** : PUKS=[x1 y1; x2 y2; x3 y3] specifies the three points
- **`mp`** : circle center
- **`r`** : circle radius

Description

This function determines the circle, which is specified by three points. Quite old geometric problem, which is used e. g. by the function nextdrei.

The strength of this function is, that it is based upon a linear equation system.

Examples

```
p=[1,1;4,3;-2,2]
[mp,r]=ponci(p)
// end of example "ponci"
```

See Also

ponbowl, nextdrei

primes - Generate list of prime numbers

Calling Sequence

```
p = primes(n)
```

Parameters

- **n** : scalar positive natural value
- **p** : row vector with prime numbers less than or equal to n

Description

A prime number is one that has no factors other than 1 and itself. This function has been imported from MatLab.

Examples

```
p=primes(100)
// end of example "primes"
```

See Also

isprime, factor

pwmratio - Pulswidth demodulation

Calling Sequence

```
[Qratio,pr,km,qpr] = pwmratio(Q)
```

Parameters

- **Q** : row vector, contains binary input sequence with {0,1} elements
- **Qratio** : PWM result, same size as Q. On the falling edges of Q the PWM result is updated
- **pr** : extracted PWM result. The length of pr is equal to the number of falling edges of Q.
- **km** : extracted period time. The length of km is equal to the falling edges of Q
- **qpr** : vector with same size as Q. On the falling edge positions of Q, qpr contains the PWM result.

Description

This function can be used to evaluate the ratio between HIGH and LOW phases of a binary input sequence. The input sequence is specified by a row vector containg elements 0 or 1.

Examples

```
x=rand(1,200,'normal');
[B,A]=butter(8,0.05);
y=mtlb_filter(B,A,x);
yb=bool2s(y>0); // Generate random binary sequence
[Qout,pr,km,qpr]=pwmratio(yb);
k0=0:1:199;
plot2d(k0,yb,style=3);
plot2d(k0,Qout,style=2);
plot2d(k0,qpr,style=7);
disp(pr);
disp(km);
```

See Also

findgrup

Used Function

findgrup

randvek - Creation of a vector with length N with numbers from 1 ... N on random positions

Calling Sequence

```
a = randvek(N)
```

Parameters

- **N** : natural positive number, length of random vector
- **a** : random vector with elements in the range [1,...,N]. Every number occurs only one time in the vector.

Description

The function uses a while loop, where the random vector is build up iteratively. Therefore, the evaluation time is not fixed.

Examples

```
a=randvek(10)
// That's it.
```

randzone - contour lines of a binary image

Calling Sequence

```
[Sumriss,csout] = randzone(S)
```

Parameters

- **S** : binary image, represented by a matrix with elements {0,1}
- **Sumriss** : binary image containing the contours of the image S.
- **csout** : vector data representing the contour lines

Description

The contour lines of areas of a matrix are calculated. The matrix Sumriss has points (1), where an area edge of S occures. When two output arguments are used, the 2nd argument csout contains vector data of the edges. With "showcs(csout)" the vector data can be displayed.

The area contours are evaluated by a combination of several binary erosions.

Examples

```
x=linspace(-%pi,%pi,200); y=x;
[X,Y]=meshgrid(x,y);
R=sqrt(X .^2 + Y .^2);
Z=sin(3 .*R);
Zbin=bool2s(Z>0.1);
// Only Binary Contour Image
Sr1=randzone(Zbin);
f=gcf();
f.color_map=graycolormap(2);
Matplot(Sr1+1);
xclick();
delete(f);
// Binary Contour Image + Contour data
[Sr2,csout]=randzone(Zbin);
showcs(csout);
```

See Also

sperob, spdilb

Used Function

sperob, spdilb, rot90

rot90 - Rotate matrix 90 degrees

Calling Sequence

```
B = rot90(A,k)
```

Parameters

- **A** : input matrix
- **k** : number of 90° rotations
- **B** : output matrix

Description

ROT90(A) is the 90 degree counterclockwise rotation of matrix A.

ROT90(A,K) is the K*90 degree rotation of A, K = +-1,+-2,...

This function has been imported from MatLab

Examples

```
A=[1,2,3;4,5,6];
B=rot90(A)
C=rot90(A,2)
D=rot90(A,3)
F=rot90(A,-1)
// End of example rot90
```

See Also

randzone

schnitt - common elements of two matrices X and Y

Calling Sequence

```
[c,Xex,Yex,Prb] = schnitt(x,y)
```

Parameters

- **x** : Set 1, specified by a vector containing positive, natural numbers.
- **y** : Set 2, specified by a vector containing positive, natural numbers. Set 1 and 2 may have different sizes.
- **c** : common elements of Set 1 and 2
- **Xex** : Elements, only available in Set 1
- **Yex** : Elements, only available in Set 2
- **Prb** : Probability vector with thre elements PRB=[P(Y<X), P(Y=X), P(Y>X)]

Description

The routine is based upon a comparison of the histograms, calculated with the SciLab function dsearch. The probabilites are derived by convolution of the histograms.

Examples

```
// Set 1
a =[12    22     5     7    38    12    30    21];
// Set 2
b =[42     6    34     7    14    39    34    26    12    25];
[c,X,Y, prb]=schnitt(a,b)
```

See Also

randvek

schnittp - crossing points of two curves

Calling Sequence

```
Q = schnittp(C1,C2)
```

Parameters

- **C1** : curve 1, defined as a complex vector
- **C2** : curve 2, defined as a complex vector
- **Q** : matrix with 3 rows describing the crossing points.

Description

This function calculates the crossing points of two curves. Therefore, curve segements by sequential points are rebuild as lines and a mathematical solution is derived. Q contains in the 1st row the crossing points, the 2nd row contains the direction vector and the last row the index p_s of curve 1, so that the crossing points can also be calculated by $C_1(p_s) + Q(3,p_s)\ (C_1(p_s+1)-C_1(p_s))$

Examples

```
z1=bezier();   // Draw Curve 1, finish with right click (twice)
z2=bezier();   // Draw Curve 2, finish with right click (twice)
// Calculating... please wait
Q=schnittp(z1,z2);
plot2d(real(z1),imag(z1),style=2);
plot2d(real(z2),imag(z2),style=3);
plot2d(real(Q(1,:)),imag(Q(1,:)),style=-3);
// Draw both curves and common points in a graph
```

See Also

meshgrid, bezier

Used Function

meshgrid

shiftmat - Matrix shifting

Calling Sequence

```
xneu = shiftmat(x,dx,dy)
```

Parameters

- **x** : matrix to be shifted
- **dx** : number of columns to the right
- **dy** : number of rows to the bottom
- **xneu** : shifted matrix

Description

Very simple function, but is used in some image processing functions, therefore a separate Sci-File has been used.

Examples

```
a=rand(5,3)
b=shiftmat(a,1,2)
// That's it
```

simpson - Numerical integration using simpson rule

Calling Sequence

```
area = simpson(y,dx)
```

Parameters

- **y** : matrix or vector, representing one or more functions (Each row is one function)
- **dx** : equidistant abscissa distance
- **area** : scalar or column vector with integration result

Description

Integration can be so simple !

Examples

```
[X,P]=meshgrid(0:(%pi/100):(2*%pi),0:0.1:1);
Y=cos(P .*X) .^2;
area=simpson(Y,%pi/100);
disp(area);
x=0:0.1:1;
y=x .^2;
area2=simpson(y,0.1);
disp(area2);
// 2 examples showing simpson
```

sirp - Sherical invariant random processes

Calling Sequence

```
[s,ABBMAT] = sirp(N,C,modusirp)
```

Parameters

- **N** : Number of samples of the random processes
- **C** : Co-Variance matrix (2 x 2) of the random processes
- **modusirp** : Modus-String. Valid entries are 'uniform' and 'normal'
- **s** : random processes, each column represents a signal
- **ABBMAT** : transformation matrix

Description

This function is used to generate two spherical invariant random processes. Herefore, the Covariance matrix between the two desired processes must be specified.

s is a matrix with 2 columns, each column represents one process. The matrix ABBMAT contains a 2x2 matrix, which can be used to generate other processes.

Examples

```
C=[0.7,0.3;0.3,0.4]
[S,abbm]=sirp(1000,C,'normal');
plot2d(S(:,1),S(:,2),style=-8);
disp(abbm);
// Generate a SIRP
x1=rand(1,200,'normal');
x2=rand(1,200,'normal');
X=[x1;x2];
Y=abbm*X;
xclick();
xbasc();
plot2d(Y(1,:),Y(2,:),style=-3);
// End of example sirp
```

spdilb - Binary Dilation of matrices

Calling Sequence

```
Qdil = spdilb(Q,mask)
```

Parameters

- **`Q`** : matrix with binary information. Elements are {0,1}
- **`mask`** : dilation window
- **`Qdil`** : output matrix, containing elements {0,1}

Description

Every element of Q, which is 1, is dilated with the mask.

This function is used e.g. in the function "randzone" and it is also a fundamental operation for many image processing functions.

Examples

```
Q=zeros(8,9);
Q(5,4)=1
mask=ones(3,3);
Qout=spdilb(Q,mask)
// End of dilation example
```

See Also

sperob, randzone

Used Function

shiftmat

sperob - Binary Erosion of matrices

Calling Sequence

```
Qero = sperob(Q,mask)
```

Parameters

- **Q** : Binary input matrix, containing elements {0,1}
- **mask** : The erosion mask
- **Qero** : The eroded binary image

Description

Sperob performs a binary erosion of the image. The erosion is specified by the matrix mask.

sperob and spdilb are fundamental functions for image processing functions (on binary images).

Examples

```
Q=zeros(8,9);
Q(6,7)=1;
Q(2:5,3:5)=ones(4,3)
mask=[0,1,0;1,1,1;0,1,0]
Qout=sperob(Q,mask)
// End of example sperob
```

See Also

spdilb, randzone

Used Function

shiftmat

spinup - row rotation of a matrix

Calling Sequence

```
y = spinup(x,a)
```

Parameters

- **x** : input matrix
- **a** : number of rows (positive number)
- **y** : output matrix

Description

Each row of the matrix X is shifted by A rows to the top. The first A rows are shifted to the back. A must be positive.

Examples

```
x=rand(5,3)
y=spinup(x,2)
// end of example spinup
```

See Also

shiftmat

timefreq - calculation of short time spectrograms of a time series

Calling Sequence

```
[spk,timeaxis,faxis] = timefreq(x,fs)
```

Parameters

- **`x`** : Time series (equidistant samples)
- **`fs`** : sampling frequency
- **`spk`** : abs value of spectrogram
- **`timeaxis`** : vector with time information
- **`faxis`** : frequency axis

Examples

```
t=(0:1:9000) ./1200;
fs=200 + 50 .*sin(2 .*%pi .*t /2);
phi=cumsum(2 .*%pi .*fs ./1200);
x=sin(phi);
[spk,tax,fax]=timefreq(x,1200);
plot3d(fax,tax,spk);
// End of example timefreq
```

See Also

welch, burg

tkblatt - evaluating the TK25-sheetnumber from longitude and lattitude

Calling Sequence

```
tknr = tkblatt(laenge,breite)
```

Parameters

- **`laenge`** : longitude in degree
- **`breite`** : lattitude in degree
- **`tknr`** : TK25 sheet number

Description

The official institution "Deutsches Landesvermessungsamt" provides maps of germany in the relation 1:25000. Each map covers a range of 4km. Whole germany is represented by 100's of these maps, each of them has a number, the TK25 number.

Examples

```
tknr=tkblatt(8.2,49.8)
// end of tkblatt
```

See Also

geo2gkk, geo2gkn

vecsort - Defragment vector data

Calling Sequence

```
csout = vecsort(csin)
```

Parameters

- **`csin`** : vector data
- **`csout`** : defragmented vector data

Description

CSIN is a contour data matrix, which is delivered for example by the function [xc,yc]=contour2di(...); cs=[xc;yc]

Normally, it is not necessary to defragment the result of contourdi. But when you have other contour data (e. g. vector data of maps), this function is quite usefull.

See Also

schnittp

welch - Welch spectrum estimation

Calling Sequence

```
[H,W] = welch(xf,N)
```

Parameters

- **xf** : Time Series as vector
- **N** : segment size. argument is optional, its default is 1024.
- **H** : The estimated spectrum
- **W** : The frequency axis with range of values [0,...,π]. π represents hereby the half of the sampling frequency.

Description

The time series is splitted with overlapping, each segment is windowed by hamming, then FFT transforms are done.

H represents not a power spectrum but an amplitude spectrum.

Examples

```
x=rand(1,10000,'normal');
hz=iir(8,'bp','butt',[0.1,0.25],[1E-5,1E-5]);
y=flts(x,hz);
[H1,W1]=welch(y,1024);
plot2d(W1 ./(2 .*%pi),H1,style=3);
xclick();
[H2,W2]=welch(y,256);
plot2d(W2 ./(2 .*%pi),H2,style=2);
// End of wlech demo
```

See Also

burg, timefreq

werktage - No. of working days between two dates

Calling Sequence

```
anzwerk = werktage(Q0,bdg,kath)
```

Parameters

- **Q0** : Date Matrix. 1st row contains start date, the 2nd the end date. The columns are sorted in the order [day,month,year]
- **bdg** : String with abbreviation of a german country (see also FEIERTAG.M), default is 'HES' for Hessen.
- **kath** : 1 for catholic regions (having more celeb days)
- **anzwerk** : No. of working days (incl. start and end date) . The dates 24.12. and 31.12 are counted with a half working day, when they are not on saturday or sunday.

Description

This function calculates the number of working days between two dates (in Germany). I am sure, you can extend the code for your countries by editing the scilab function "feiertag".

Examples

```
// number of working days in 2006 in Germany
// takes a little bit time, sorry.
Q=[1,1,2006;31,12,2006]
aw=werktage(Q);
disp(aw);
// end of werktage demo
```

See Also

feiertag, dday

Used Function

feiertag

workdays - create string table with information about working days

Calling Sequence

```
S = workdays(j,bdg,kath)
```

Parameters

- **j** : vector with year numbers
- **bdg** : abbr. for german countries, default is 'hes' for Hessen
- **kath** : scalar (=1 for catholic areas)
- **S** : string matrix representing a chart.

Description

This function uses the function "werktage". The result is a chart. Each column represents one year, the rows a showing the months of the year with the number of working days.

The chart contains a heading line and a final sum row, therefore S has always 14 rows.

Examples

```
// calculate the number of
// working days for the year
// 2005, ..., 2008 in Hessen (Germany)
// Please wait a little bit...
S=workdays(2005:2008);
```

See Also

werktage, ostern

Used Function

werktage

Stichwortverzeichnis