MarcumQ

MarcumQ(*m*, *a*, *b*)

The MarcumQ function returns the generalized Q-function defined by the integral

$$Q_m(a,b) = \int_b^\infty u\left(\frac{u}{a}\right)^{m-1} \exp\left(-\frac{(a^2+u^2)}{2}\right) I_{m-1}(au)\,du$$

where I_k is the modified Bessel function of the first kind and order k.

Its applications have been primarily in the fields of communication and detection theory. However, an interesting interpretation of its result with m=1 and appropriate parameter scaling is the fractional power of a two-dimensional circular Gaussian function within a displaced circular aperture.

Depending on the input arguments, the MarcumQ function may be computationally intensive but you can abort the calculation at any time.

References

Cantrell, P.E., and A.K. Ojha, Comparison of Generalized Q-Function Algorithms, *IEEE Transactions on Information Theory*, *IT-33*, 591-596, 1987.

Simon, M. K., A New Twist on the Marcum Q-Function and Its Application, *IEEE Communications Letters*, 3, 39-41, 1998.

MarkPerfTestTime

MarkPerfTestTime *idval*

Use the MarkPerfTestTime operation for performance testing of user-defined functions in conjunction with `SetIgorOption DebugTimer`. When used between `SetIgorOption DebugTimer, Start` and `SetIgorOption DebugTimer, Stop`, MarkPerfTestTime stores the ID value and the time of the call in a buffer. When `SetIgorOption DebugTimer, Stop` is called the contents of the buffer are dumped to a pair of waves: W_DebugTimerIDs will contain the ID values and W_DebugTimerVals will contain the corresponding times of the calls relative to the very first call. The timings use the same high precision mechanism as the StartMSTimer and StopMSTimer calls.

By default, `SetIgorOption DebugTimer, Start` allocates a buffer for up to 10000 entries. You can allocate a different sized buffer using `SetIgorOption DebugTimer, Start=bufsize`.

See Also

SetIgorOption, **StartMSTimer**, and **StopMSTimer**.

Additional documentation can be found in an example experiment, PerformanceTesting.pxp, and a WaveMetrics procedure file, PerformanceTestReport.ipf.

MatrixBalance

MatrixBalance [flags] *srcWave*

The MatrixBalance operation permutes and/or scales an NxN real or complex matrix to obtain a similar matrix that is more stable for subsequent calculations such as eigenvalues and eigenvectors. Balancing a matrix is useful when some matrix elements are much smaller than others.

MatrixBalance saves the resulting matrix in the wave specified by the /DSTM flag. It creates a secondary output wave specified by the /DSTS flag containing permutation and scaling information.

MatrixBalance was added in Igor Pro 9.00.

Parameters

srcWave must be single-precision or double-precision floating point, real or complex, and must contain no NaNs. MatrixBalance returns an error if these conditions are not met.

Flags

/DSTM=*dest*	Specifies the destination wave for the balanced output matrix. If you omit /DSTM, the output is saved in M_Balanced in the current data folder.

/DSTS=*dest* Specifies the destination wave for the scaling array. If you omit /DSTS, the scaling wave is saved as W_Scale in the current data folder.

/FREE Creates free destination waves when they are are specified via /DSTM or /DSTS.

/J=*job* *job* is one of the following letters:

- N *srcWave* is not permuted or scaled.
- P *srcWave* is permuted but not scaled.
- S *srcWave* is scaled but not permuted. The scaling applies a diagonal similarity transformation to make the norms of the various columns close to each other.
- B *srcWave* is both scaled and permuted (default).

/Z Do not report any errors. V_flag will be set to 0 if successful or to an error code. V_min and V_max will be set to NaN in case of an error.

Details

Matrix balancing is usually called internally by LAPACK routines when there is large variation in the magnitude of matrix elements. The /J flag supports the granularity of applying only permutation, only scaling or both.

After balancing there will be a block diagonal form that could typically be handled using Schur decomposition (see MatrixSchur). The rows/columns corresponding to this block diagonal are saved in zero based V_min and V_max. The scaling wave W_scale[i] (for i<V_min) contains the index of the row/column that was interchanged with row/col i. For i>=V_min the scaling wave contains the scaling factor use to balance row and column i.

When you balance *srcWave* using this operation the resulting eigenvectors belong to the balanced matrix. Use **MatrixReverseBalance** to obtain the eigenvectors of *srcWave*.

When solving for the Schur vectors of *srcWave* after using /J=P for permuting only, use **MatrixReverseBalance** to transform the vectors. If you are solving for the Schur vectors do not use either /J=S or /J=B because that balancing transformation is not orthogonal.

Output Variables

V_Flag Set to zero when the operation succeeds. Otherwise, when V_flag is positive the value is a standard error code. When V_flag is negative it is an indication of a wrong input parameter.

V_min Set to zero if /J=N or /J=S.

V_max Set to N-1 (where N is the number of rows or columns) if /J=N or /J=S.

When /J=B or /J=P the matrix M_Balanced[i][j]=0 if i>j and j=0...V_min-2 or i=V_max,...N-1.

Examples

```
Function DemoMatrixBalance1()
   // Generate random unbalanced data
   Make/D/O/N=(7,7) srcWave=p+10*q+enoise(5)
   MatrixBalance srcWave
   Wave M_Balanced
   Wave W_Scale

   MatrixOP/O/FREE matD=diagonal(W_Scale)
   MatrixOP/O/FREE matDInv=inv(matD)

   // Check that W_Scale produces the correct balanced matrix
   MatrixOP/O/FREE matProd=matDInv x srcWave x matD

   // Compare with M_Balanced
   MatrixOP/O/FREE/P=1 aa=sum(abs(M_Balanced-matProd))
End
Function DemoMatrixBalance2()
   Make/O/N=(3,3) mat
```