/TET=*nTerms* Sets the number of terms in the Taylor expansion. Use /TET to set the number of terms and bypass the default error estimate, which is estimated from the approximate error value (/AERR).

/Z No error reporting.

Details

The discrete Gauss transform can be computed as a direct sum. An exact calculation is practical only for moderate number of sources and observation points and for low spatial dimensionality. With increasing dimensionality and increasing number of sources it is more efficient to take advantage of some properties of the Gaussian function. The FastGaussianTransform operation does so in two ways: It first arranges the sources in N-dimensional spatial clusters so that it is not necessary to compute the contributions of all source points that belong to remote clusters (see **FPClustering**). The second component of the algorithm is an approximation that factorizes the sum into a factor that depends only on source points and a factor that depends only on observation points. The factor that depends only on source points is computed only once while the factor that depends on observation points is evaluated once for each observation point.

The trade-off between computation efficiency and accuracy can be adjusted using multiple parameters. By default, the operation calculates the number of terms it needs to use in the Taylor expansion of the Gaussian. You can modify the default approximate error value using /AERR or you can directly set the number of terms in the expansion using /TET.

FastGaussianTransform supports calculations in dimensions that may exceed the maximum allowed wave dimensionality. *srcLocationsWave* must be a 2D, real-valued single- or double-precision wave in which each row corresponds to a single source position and columns represent the components in each dimension (e.g., a triplet wave would represent 3D source locations). *srcWeightsWave* must have the same number of rows as *srcLocationsWave* and it must be a real-valued single- or double-precision wave. In most applications *srcWeightsWave* will have a single column so that the output ***G*** will be scalar. However, if *srcWeightsWave* has multiple columns than ***G*** is a vector. This can be handy if you need to test multiple sets of coefficients at one time. If you specify observation points using /OUTW then *locWave* must have the same number of columns as *srcLocationsWave* (the number of rows in the output is arbitrary). The operation does not support wave scaling.

See Also

The **CWT**, **FFT**, **ImageInterpolate**, **Loess**, and **FPClustering** operations.

References

Yang, C., R. Duraiswami, and L. Davis, Efficient Kernel Machines Using the Improved Fast Gauss Transform, *Advances in Neural Information Processing Systems 16*, 2004.

FastOp

FastOp *destWave* = *prod1* [**± *prod2*** [**± *prod3***]]
FastOp *destWave* += *prod1* [**± *prod2***]

The FastOp operation can be used to get improved speed out of certain wave assignment statements. The syntax was designed so that you can simply insert `FastOp` in front of any wave assignment statement that meets the syntax requirements. The += syntax was added in Igor Pro 9.00.

Parameters

destWave An existing destination wave for the assignment expression. An error will be reported at runtime if the waves are not all the same length or number type.

prod1, *prod2*, *prod3* Products with the following formats:

constexpr wave1 * wave2*

or

*constexpr * wave1 / wave2*

constexpr may be a literal numeric constant or a constant expression in parentheses. Such expressions are evaluated only once.

Any component in a prod expression may be omitted.

Flags

/C — The /C flag is obsolete in Igor Pro 9.00 and later and is ignored. For floating point waves, the type of expressions, real or complex, is determined by the type of the destination wave.

In Igor Pro 8 and before /C is required when using a complex destination wave.

FastOp supports real and complex for floating point waves but only real for integer waves.

Details

FastOp supports real and complex for floating point waves. It supports real only for integer waves.

If your waves are complex, make sure to use WAVE/C when declaring wave references in user-defined functions.

Certain combinations, listed in the following table, are evaluated using faster optimized code rather than more general but slower generic code. SP means "single-precision floating point" and DP means "double-precision floating point".

Statement	Optimized Data Types
dest = 0	All
dest = waveA	All
*dest += C0 * waveA*	SP and DP, real and complex
*dest = dest + C0 * waveA*	SP and DP, real and complex
*dest = dest + waveA * C0*	SP and DP, real and complex
dest = C0	SP and DP and integer, real only
dest = waveA +C1	SP and DP and integer, real only
*dest = C0 * waveA + C1*	SP and DP, real only
dest = waveA + waveB + C2	Integer only, real only

In the statements above, pluses may be minuses and the trailing constant (*C0, C1, C2*) may be omitted.

Note: Except for the optimized cases listed above which use integer calculations for integer waves, integer results are evaluated using double precision intermediate values. This limits precision for 64-bit integer waves to 53 bits.

Speedups generally range from 10 to 40 times faster than the equivalent statement with FastOp removed. The speedup is dependent on the computer and on the length of the waves with the greatest improvement occurring for waves with 1000 to 100,000 points.

Examples

Valid expressions:

```
FastOp dest = 3
FastOp dest = waveA + waveB
FastOp dest = 0.5*waveA + 0.5*waveB
FastOp dest = waveA*waveB
FastOp dest = (2*3)*waveA + 6
FastOp dest = (locvar)*waveA
FastOp dest += waveA + waveB
```

Expressions that are **not** valid:

```
FastOp dest = 3*4
FastOp dest = (waveA + waveB)
FastOp dest = waveA*0.5 + 0.5*waveB
FastOp dest = waveA*waveB/2
FastOp dest = 2*3*waveA + 6
FastOp dest = locvar*waveA
FastOp dest += waveA + waveB + waveC
```