Quick guide for converting old fff code (GSL dependent) into new code

Vectors

Essentially, this is the same API as GSL up to the prefix (gsl_ is to be replaced with fff_). The basic routines have been rewritten from scratch, while the BLAS routines have been wrapped around the standard Fortran BLAS API (as opposed to CBLAS). There are only three exceptions (in red in the following table).

For more details, see: fff_vector.h, fff_blas.h.

old	new	remarks
`gsl_vector* x`	`fff_vector* x`	
`x = gsl_vector_alloc(n)`	`x = fff_vector_new(n)`	*new* rather than *alloc* for consistency with other fff structures
`gsl_vector_free(x)`	`fff_vector_delete(x)`	same remark as above
`gsl_vector_set(x, i, a)`	`fff_vector_set(x, i, a)`	
`gsl_vector_set_all(x, a)`	`fff_vector_set_all(x, a)`	
`gsl_vector_set_zero(x)`	`fff_vector_set_all(x, 0)`	
`gsl_vector_memcpy(y, x)`	`fff_vector_memcpy(y, x)`	
`gsl_vector_add(y, x)`	`fff_vector_add(y, x)`	similarly for sub, mul, div, scale, ...
`gsl_vector_get(x, i)`	`fff_vector_get(x, i)`	
`gsl_blas_dgemv(CblasNoTrans,a,x,b,y)`	`fff_blas_dgemv(CblasNoTrans,a,x,b,y)`	similarly for *all* gsl_blas functions execpt the following
`gsl_blas_ddot(x,y,&v)`	`v = fff_blas_ddot(x,y)`	

Matrices

No major modification regarding low-level routines, see fff_matrix.h for details. For more sophisticated linear algebra (LU, QR, SVD, Cholesky decompositions), use the wrapper around LAPACK (fff_lapack.h) whose API is completely different from GSL.

old	new	remarks
`gsl_matrix* A`	`fff_matrix* A`	
`A = gsl_matrix_alloc(nr,nc)`	`A = fff_matrix_new(nr,nc)`	new rather than alloc for consistency with other fff structures
`gsl_matrix_free(A)`	`fff_matrix_delete(A)`	Same remark as above
`gsl_matrix_get(A,i,j)`	`fff_matrix_get(A,i,j)`	
`gsl_matrix_set(A,i,j,a)`	`fff_matrix_set(A,i,j,a)`	
`gsl_matrix_get_row(x,A,i)`	`fff_matrix_get_row(x,A,i)`	
`fff_matrix_get_col(x,A,j)`	`fff_matrix_get_col(x,A,j)`	
`gsl_matrix_set_row(A,i,x)`	`fff_matrix_set_row(A,i,x)`	
`gsl_matrix_set_col(A,j,x)`	`fff_matrix_set_col(A,j,x)`	
?	`fff_matrix_get_diag(x,A)`	
?	`fff_matrix_set_diag(A,x)`	

Vector & matrix views

We haven't implemented a specific type for vector and matrix views. Views directly output a fff_vector or a fff_matrix. See: fff_matrix.h.

old	new	remarks
`gsl_matrix* A`	`fff_matrix* A`	
`gsl_vector_view v;` `v = gsl_matrix_row(A,i);`	`v = fff_vector;` `v = fff_matrix_row(A,i);`	You can then pass &v to a function rather than &v.vector
`gsl_vector_view v;` `v = gsl_matrix_col(A,i);`	`v = fff_vector;` `v = fff_matrix_col(A,i);`	

`gsl_vector_view v;`
`v =`
`gsl_matrix_diagonal(A,i);`
	`v = fff_vector;`
	`v = fff_matrix_diag(A);`	

`gsl_matrix_view B;`
`B =`
`gsl_matrix_submatrix(k1,k2,n1,n2);`
	`fff_matrix B;`
	`B =`
	`fff_matrix_block(k1,n1,k2,n2);`
		Beware: input arguments are not in the same order

Non-double arrays

The former *fff_image* type has been renamed *fff_array* and a few macros have been added so as to simplify its API. I suggest using *fff_array* rather than re-implementing GSL's manual templates such as gsl_vector_long and gsl_matrix_long, unless computational efficiency collapses dramatically.

old	new	remarks
`gsl_vector_long* x =` `gsl_vector_long_alloc(n)`	`fff_array* x =` `fff_array_new1d(FFF_LONG, n)`	fff_array_new1d is a macro for fff_array_new(FFF_LONG,n,0,0,0)
`gsl_vector_long_free(x)`	`fff_array_delete(x)`	
`long int v =` `gsl_vector_long_get(x,i)`	`long int v = (long int)` `fff_array_get1d(x,i)`	Macro for fff_array_get(x,i,0,0,0), which returns a double
`gsl_vector_long_set(x,i,a)`	`fff_array_set1d(x,i,a)`	Macro for fff_array_set(x,i,0,0,0,a)
`gsl_vector_long_set_all(x,a)`	`fff_array_set_all(x,a)`	
`gsl_matrix_long* A =` `gsl_matrix_long_alloc(p,q)`	`fff_array* A =` `fff_array_new2d(FFF_LONG,p,q)`	
`gsl_matrix_long_free(A)`	`fff_array_delete(A)`	
`long int v =` `gsl_matrix_long_get(A,i,j)`	`long int v = (long int)` `fff_array_get2d(A,i,j)`	
`gsl_matrix_long_set(A,i,j,a)`	`fff_array_set2d(A,i,j,a)`	
`gsl_matrix_long_set_all(A,a)`	`fff_array_set_all(A,a)`	