NUFFT

Version 1.2

FORTRAN 77, FORTRAN 90

Beta release

January 20, 2009

User's guide

Contents

1 Introduction

This brief manual explains how to use the NUFFT suite for one, two and three dimensions. In each dimension, we distinguish between three types of problems:

1. transforming nonequispaced "function values" to integer frequencies,
2. evaluating a Fourier expansion at nonequispaced locations, and
3. transforming nonequispaced "function values" to non-integer frequencies.

More precisely, in one dimension, these formulas take the form

$$f(k) = \sum_{j=0}^{N-1} c_j \, e^{\pm ik \cdot p_j} \qquad \text{1D Type 1}$$

$$c_j = \sum_{k=-\frac{M}{2}}^{\frac{M}{2}-1} f(k) \, e^{\pm ip_j k}, \qquad \text{1D Type 2}$$

$$f(s_k) = \sum_{j=0}^{N-1} c_j \, e^{\pm is_k \cdot p_j} \ . \qquad \text{1D Type 3}$$

For transformations of types 1 and 2, we assume $p_j \in [-\pi, \pi]$. In two dimensions, these formulas take the form

$$f(k_1, k_2) = \sum_{j=0}^{N-1} c_j \, e^{\pm i(k_1,k_2) \cdot \mathbf{p}_j} \qquad \text{2D Type 1}$$

$$c_j = \sum_{k_1=-\frac{M_1}{2}}^{\frac{M_1}{2}-1} \sum_{k_2=-\frac{M_2}{2}}^{\frac{M_2}{2}-1} f(k_1, k_2) \, e^{\pm i\mathbf{p}_j \cdot (k_1,k_2)}, \qquad \text{2D Type 2}$$

$$f(\mathbf{s}_k) = \sum_{j=0}^{N-1} c_j \, e^{\pm i\mathbf{s}_k \cdot \mathbf{p}_j} \qquad \text{2D Type 3}$$

respectively. For transformations of types 1 and 2, we assume $\mathbf{p}_j \in [-\pi, \pi] \times [-\pi, \pi]$. For transformations of type 3, we assume only that $\mathbf{s}_k \in \mathbf{R}^2$. In three dimensions, these formulas take the form

$$f(k_1, k_2, k_3) = \sum_{j=0}^{N-1} c_j \, e^{\pm i(k_1,k_2,k_3) \cdot \mathbf{p}_j} \qquad \text{3D Type 1}$$

$$c_j = \sum_{k_1=-\frac{M_1}{2}}^{\frac{M_1}{2}-1} \sum_{k_2=-\frac{M_2}{2}}^{\frac{M_2}{2}-1} \sum_{k_3=-\frac{M_3}{2}}^{\frac{M_3}{2}-1} f(k_1, k_2, k_3) \, e^{\pm i\mathbf{p}_j \cdot (k_1,k_2,k_3)}, \qquad \text{3D Type 2}$$

$$f(\mathbf{s}_k) = \sum_{j=0}^{N-1} c_j \, e^{\pm i\mathbf{s}_k \cdot \mathbf{p}_j} \qquad \text{3D Type 3}\,,$$

respectively. For transformations of types 1 and 2, we assume $\mathbf{p}_j \in [-\pi, \pi]^3$. For transformations of type 3, we assume only that $\mathbf{s}_k \in \mathbf{R}^3$.

Transformations of type 3 require the most sophistication on the part of the user. Poorly considered choices of the ranges of $\mathbf{s}_k$ and $\mathbf{p}_j$ can have significant impact on performance.

The articles on nonuniform FFTs on which this code is based are

1. L. Greengard and J.-Y. Lee, *Accelerating the Nonuniform Fast Fourier Transform*, SIAM Review, **46**, 443–454 (2004).

2. J.-Y. Lee and L. Greengard, *The Nonuniform FFT of Type 3 and its Applications*, J. Comput. Phys., **206**, 1–5 (2005).

Convention concerning source locations

In the type 1 and 2 transforms, the SIAM Review paper used the convention that points lie in $[0, 2\pi]$. In the NUFFT library, they are assumed to lie in $[-\pi, \pi]$.

Performance

These codes are not optimized for performance, although they do achieve the desired $O(N \log N)$ scaling. They are intended to be readable by computational scientists who are familiar with the underlying techniques so that they can be modified as needed. In particular,

1. We initialize the FFT on every call.

2. We do not precompute the exponentials involved in "fast Gaussian gridding".

3. We do not block structure the code so that irregularly placed points are interpolated ("gridded") in a cache-aware fashion.

4. We have not used the state of the art FFTW package (`http://www.fftw.org`). As written, the NUFFT calls the FFT routines available from Netlib in the `bihar` repository (a double precision version of `fftpack`).

Acknowledgements

This work was supported in part by the Applied Mathematical Sciences Program of the U.S. Department of Energy under contract DEFGO288ER25053 and in part by the Korea Research Foundation under grant 2002-015-CP0044.

2 The DIRFT*d1 Subroutines

The DIRFT*d1 subroutines compute the nonuniform FFTs of type 1 directly. The calling sequences for the double precision codes are

```
call dirft1d1(nj, xj, cj, iflag, m1, fk)
call dirft2d1(nj, xj, yj, cj, iflag, m1, m2, fk)
call dirft3d1(nj, xj, yj, zj, cj, iflag, m1, m2, m3, fk)
```

where
$p_j =$ `xj(j)` in the 1D case,
$\mathbf{p}_j =$ `(xj(j),yj(j))` in the 2D case,
$\mathbf{p}_j =$ `(xj(j),yj(j),zj(j))` in the 3D case (see formulas on p. 2).

2.1 DIRFT*d1 input parameters

`nj:` `integer` : number of sources

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth source.

`cj(nj):` `complex *16` : function value cj(j) at jth source

`iflag:` `integer` : If (`iflag` ≥ 0) the exponential with positive sign is used in the transformation (see p. 2). If (`iflag` < 0) the exponential with negative sign is used in the transformation.

`m1, m2, m3:` `integer` : dimensions of Fourier coefficient array

2.2 DIRFT*d1 output parameters

`fk(m1)| fk(m1,m2) | fk(m1,m2,m3):` `complex *16` : the computed Fourier coefficients in `1D|2D|3D`.

3 The DIRFT*d2 Subroutines

The DIRFT*d2 subroutines compute the nonuniform FFTs of type 2 directly. The calling sequences for the double precision codes are

```
call dirft1d2(nj, xj, cj, iflag, eps, m1, fk)
call dirft2d2(nj, xj, yj, cj, iflag, eps, m1, m2, fk)
call dirft3d2(nj, xj, yj, zj, cj, iflag, eps, m1, m2, m3, fk)
```

where
$p_j =$ `xj(j)` in the 1D case,
$\mathbf{p}_j =$ `(xj(j),yj(j))` in the 2D case and
$\mathbf{p}_j =$ `(xj(j),yj(j),zj(j))` in the 3D case (see formulas on p. 2).

3.1 DIRFT*d2 input parameters

`nj:` `integer` : number of evaluation points

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth evaluation point.

`fk(m1)| fk(m1,m2) | fk(m1,m2,m3):` `complex *16` : the given Fourier coefficients in `1D|2D|3D`.

`iflag:` `integer` : If (`iflag` ≥ 0) the exponential with positive sign is used in the transformation (see p. 2). If (`iflag` < 0) the exponential with negative sign is used in the transformation.

`m1, m2, m3:` `integer` : dimensions of Fourier coefficient array

3.2 DIRFT*d2 output parameters

`cj(nj):` `complex *16` : transform value at jth target

4 The DIRFT*d3 Subroutines

The DIRFT*d3 subroutines compute the nonuniform FFTs of type 3 directly. The calling sequences for the double precision codes are

```
call dirft1d3(nj, xj, cj, iflag, nk, sk, fk)
call dirft2d3(nj, xj, yj, cj, iflag, nk, sk, tk, fk)
call dirft3d3(nj, xj, yj, zj, cj, iflag, nk, sk, tk, uk, fk)
```

where
$p_j = \texttt{xj(j)}, s_k = \texttt{sk(k)}$ in the 1D case,
$\mathbf{p}_j = (\texttt{xj(j)}, \texttt{yj(j)}), \mathbf{s}_k = (\texttt{sk(k)}, \texttt{tk(k)})$ in the 2D case and
$\mathbf{p}_j = (\texttt{xj(j)}, \texttt{yj(j)}, \texttt{zj(j)}), \mathbf{s}_k = (\texttt{sk(k)}, \texttt{tk(k)}, \texttt{uk(k)})$ in the 3D case
(see formulas on p. 2).

4.1 DIRFT*d3 input parameters

`nj:` `integer` : number of source points

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth source point.

`cj(nj):` `complex *16` : coefficient value at jth source point

`iflag:` `integer` : If ($\texttt{iflag} \geq 0$) the exponential with positive sign is used in the transformation (see p. 2). If ($\texttt{iflag} < 0$) the exponential with negative sign is used in the transformation.

`nk:` `integer` : number of transform outputs

`sk(nk), tk(nk), uk(nk):` `real *8` : locations of transform outputs.

4.2 DIRFT*d3 output parameters

`fk(nk):` `complex *16` : values of transform outputs.

5 The NUFFT*d1 Subroutines

The NUFFT*d1f90 (Fortran 90) and NUFFT*d1 (Fortran 77) subroutines implement the type 1 transformations described in section 1. The calling sequences for the double precision codes are

```
call nufft1d1f90(nj,xj,cj,iflag,eps,m1,fk,ier)
call nufft2d1f90(nj,xj,yj,cj,iflag,eps,m1,m2,fk,ier)
call nufft3d1f90(nj,xj,yj,zj,cj,iflag,eps,m1,m2,m3,fk,ier)

call nufft1d1(nj,xj,cj,iflag,eps,m1,fk,fw,lw,lused,ier)
call nufft2d1(nj,xj,yj,cj,iflag,eps,m1,m2,fk,fw,lw,lused,ier)
call nufft3d1(nj,xj,yj,zj,cj,iflag,eps,m1,m2,m3,fk,fw,lw,lused,ier)
```

where
$p_j =$ `xj(j)` in the 1D case,
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`) in the 2D case and
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`, `zj(j)`) in the 3D case (see formulas on p. 2).

5.1 NUFFT*d1 input parameters

`nj:` `integer` : number of sources

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth source.

`cj(nj):` `complex *16` : function value cj(j) at jth source

`iflag:` `integer` : If (`iflag` ≥ 0) the exponential with positive sign is used in the transformation (see p. 2). If (`iflag` < 0) the exponential with negative sign is used in the transformation.

`eps:` `real *8` : User defined tolerance in range $[1.0d-1, 1.0d-13]$.

`m1, m2, m3:` `integer` : dimensions of Fourier coefficient array

`fw(0:lw):` `real *8` : workspace of length `lw`. [**F77 only**].

`lw` $\approx$ `20*m1 + 100` in 1D.
`lw` $\approx$ `20*m1*m2+ 10*max(m1,m2) + 100` in 2D.
`lw` $\approx$ `60*m1*m2*m3+ 10*max(m1,m2,m3) + 100` in 3D.

5.2 NUFFT*d1 output parameters

`fk(m1)| fk(m1,m2) | fk(m1,m2,m3):` `complex *16` : the computed Fourier coefficients in `1D|2D|3D`.

`ier:` `integer` : error return code.
ier = 0 for normal execution.
ier = 1 if eps is outside the allowed range.
ier = 2 if lw is not sufficiently large [**F77 only**].

`lused:` `integer` : length of workspace used [**F77 only**].

6 The NUFFT*d2 Subroutines

The NUFFT*d2f90 (Fortran 90) and NUFFT*d2 (Fortran 77) subroutines implement the type 2 transformations described in section 1. The calling sequences for the double precision codes are

```
call nufft1d2f90(nj,xj,cj,iflag,eps,m1,fk,ier)
call nufft2d2f90(nj,xj,yj,cj,iflag,eps,m1,m2,fk,ier)
call nufft3d2f90(nj,xj,yj,zj,cj,iflag,eps,m1,m2,m3,fk,ier)

call nufft1d2(nj,xj,cj,iflag,eps,m1,fk,fw,lw,lused,ier)
call nufft2d2(nj,xj,yj,cj,iflag,eps,m1,m2,fk,fw,lw,lused,ier)
call nufft3d2(nj,xj,yj,zj,cj,iflag,eps,m1,m2,m3,fk,fw,lw,lused,ier)
```

where
$p_j =$ `xj(j)` in the 1D case,
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`) in the 2D case and
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`, `zj(j)`) in the 3D case (see formulas on p. 2).

6.1 NUFFT*d2 input parameters

`nj:` `integer` : number of evaluation points

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth evaluation point.

`fk(m1)| fk(m1,m2) | fk(m1,m2,m3):` `complex *16` : the given Fourier coefficients in `1D|2D|3D`.

`iflag:` `integer` : If (`iflag` ≥ 0) the exponential with positive sign is used in the transformation (see p. 2). If (`iflag` < 0) the exponential with negative sign is used in the transformation.

`eps:` `real *8` : User defined tolerance in range $[1.0d-1, 1.0d-13]$.

`m1, m2, m3:` `integer` : dimensions of Fourier coefficient array

`fw(0:lw):` `real *8` : workspace of length `lw`. [**F77 only**].

`lw` $\approx$ `20*m1 + 100` in 1D.
`lw` $\approx$ `20*m1*m2+ 10*max(m1,m2) + 100` in 2D.
`lw` $\approx$ `60*m1*m2*m3+ 10*max(m1,m2,m3) + 100` in 3D.

6.2 NUFFT*d2 output parameters

`cj(nj)`: `complex *16` : transform value at jth target

`ier`: `integer` : error return code.
ier = 0 for normal execution.
ier = 1 if eps is outside the allowed range.
ier = 2 if lw is not sufficiently large [**F77 only**].

`lused`: `integer` : length of workspace used [**F77 only**].

7 The NUFFT*d3 Subroutines

The NUFFT*d3f90 (Fortran 90) and NUFFT*d3 (Fortran 77) subroutines implement the type 3 transformations described in section 1. The calling sequences for the double precision codes are

```
call nufft1d3f90(nj,xj,cj,iflag,eps,nk,sk,fk,ier)
call nufft2d3f90(nj,xj,yj,cj,iflag,eps,nk,sk,tk,fk,ier)
call nufft3d3f90(nj,xj,yj,zj,cj,iflag,eps,nk,sk,tk,uk,fk,ier)

call nufft1d3(nj,xj,cj,iflag,eps,nk,sk,fk,fw,lw,lused,ier)
call nufft2d3(nj,xj,yj,cj,iflag,eps,nk,sk,tk,fk,fw,lw,lused,ier)
call nufft3d3(nj,xj,yj,zj,cj,iflag,eps,nk,sk,tk,uk,fk,fw,lw,lused,ier)
```

where
$p_j =$ `xj(j)`, $s_k =$ `sk(k)` in the 1D case,
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`), $\mathbf{s}_k =$ (`sk(k)`, `tk(k)`) in the 2D case and
$\mathbf{p}_j =$ (`xj(j)`, `yj(j)`, `zj(j)`), $\mathbf{s}_k =$ (`sk(k)`, `tk(k)`, `uk(k)`) in the 3D case
(see formulas on p. 2).

7.1 NUFFT*d3 input parameters

`nj:` `integer` : number of source points

`xj(nj), yj(nj), zj(nj):` `real *8` : coordinates of jth source point.

`cj(nj):` `complex *16` : coefficient value at jth source point

`iflag:` `integer` : If (`iflag` ≥ 0) the exponential with positive sign is used in the transformation (see p. 2). If (`iflag` < 0) the exponential with negative sign is used in the transformation.

`eps:` `real *8` : User defined tolerance in range $[1.0d-1, 1.0d-13]$.

`nk:` `integer` : number of transform outputs

`sk(nk), tk(nk), uk(nk):` `real *8` : locations of transform outputs.

`fw(0:lw):` `real *8` : workspace of length `lw`. [**F77 only**].

`lw:` `integer` :
Let `SR=max(sk(k))-min(sk(k))`, `XR=max(sk(k))-min(sk(k))`,
`TR=max(tk(k))-min(tk(k))`, `YR=max(yj(k))-min(yj(k))`,
`UR=max(uk(k))-min(uk(k))`, `ZR=max(zj(k))-min(zj(k))`.

`lw` $\approx$ `12*SR*XR+500` in 1D,
`lw` $\approx$ `18*SR*XR*TR*YR + 10*max(SR*XR,TR*XR)+500` in 2D,
`lw` $\approx$ `55*SR*XR*TR*YR*UR*ZR+10*max(SR*XR,TR*XR,UR*ZR)+500` in 3D,

Remark: `lw` is actually precision-dependent, so the preceding is an overestimate for low precision and an underestimate for high precision. The next release will include a more accurate estimator.

7.2 NUFFT*d3 output parameters

`fk(nk)`: `complex *16` : values of transform outputs.

`ier`: `integer` : error return code.
ier = 0 for normal execution.
ier = 1 if eps is outside the allowed range.
ier = 2 if lw is not sufficiently large [**F77 only**].

`lused`: `integer` : length of workspace used [**F77 only**].