Fourier integral approximation with FFT

Janis Erdmanis
akels14@gmail.com

June 12, 2014

1 The problem and its solution

Recently I had to compute number of integrals for multiple p in the form of:

$$\hat{f}(p) = \int_{-\infty}^{+\infty} e^{-ixp} f(x) dx \tag{1}$$

which is just continious Fourier transform. The performance with Gaus-Kronrod quadrature was quite acceptable in the limit where $p \to 0$, but in most cases it took too long. Doing some research on it I found [fourint] where the Fourier integral is approximated with trapezodial rule and given for FFT for computing sums.

The requrement on function $f(x)$ is to be localised in some interval $[a, b]$ and also it needs to be bounded or $|f(x)| \leq M$ which allows to rewrite (1):

$$\hat{f}(p) = \int_{a}^{b} e^{-ixp} f(x) dx \tag{2}$$

where introduced error of approximation will be estimated later. For simplicity we can change integration variables $x = (b - a)y + a$:

$$\begin{aligned}
\hat{f}(p) &= \int_0^1 e^{-i(b-a)yp} e^{-iap} f[(b-a)y + a](b-a) dy && (3)\\
&= (b-a) M e^{-iap} \int_0^1 e^{-i(b-a)yp} \frac{f[(b-a)y + a]}{M} dy && (4)\\
&= (b-a) M e^{-iap} \hat{f}'[(b-a)p] && (5)
\end{aligned}$$

Therefore the general problem is reduced to computation of:

$$\hat{f}'(p) = \int_0^1 e^{-ixp} f'(x) dx \tag{6}$$

where $f'(x) = f[(b-a)y + a]/M$.

At the next step we are going to introduce the trapezodial approximation where by using $f(a) = f(b) = 0$ we obtain:

$$\hat{f}'(p) = \int_0^{(N-1)\Delta x} e^{-ixp}\Theta(1-x)f'(x)dx \tag{7}$$

$$= \sum_{n=0}^{N-1} e^{-ipn\Delta x}\Theta(1-x_n)f'(x_n)\Delta x \tag{8}$$

$$= \frac{1}{N_0-1}\sum_{n=0}^{N-1} e^{-inp/(N_0-1)}\Theta(1-x_n)f'(x_n) \tag{9}$$

where $\Delta x = 1/(N_0-1)$, $x_n = n/(N_0-1)$ and $\Theta(x)$ is Heaviside step function. Now we are specifieng the values of transformed variable p for which the integral is calculated[1]:

$$p_m = 2\pi(N_0-1)\left(\frac{m}{N}-\frac{1}{2}\right) \tag{10}$$

which after putting in (9) one obtains:

$$\hat{f}'(p_m) = \frac{1}{N_0-1}\sum_{n=0}^{N-1} e^{-2\pi inm/N}e^{-\pi in}\Theta(1-x_n)f'(x_n) \tag{11}$$

Now it is appearent that the sum can be expressed as discrete Fourier transform giving:

$$\hat{f}'(p_m) = \frac{1}{N_0-1}FFT\{(-1)^n\Theta(1-x_n)f'(x_n)\}_m \tag{12}$$

2 Error estimations

First error was introduced considering the $f(x)$ to be localised with what we rewrited (1) to (2) where for exact calculation we would write:

$$\int_{-\infty}^{+\infty} e^{-ixp}f(x)dx = \int_a^b e^{-ixp}f(x)dx + \int_b^{+\infty} e^{-ixp}f(x)dx + \int_{-\infty}^a e^{-ixp}f(x)dx \tag{13}$$

Since $e^{-ixp} \leq 1$ then the error can be estimated:

$$error = \left|\int_{-\infty}^{+\infty} e^{-ixp}f(x)dx - \int_a^b e^{-ixp}f(x)dx\right| \leq \int_b^{+\infty}|f(x)|dx + \int_{-\infty}^a |f(x)|dx \tag{14}$$

where the last integrals could be done numerically.

The second error comes from approximating integral with trapezidual rule

[1] It is possible to shift p_m in any region however for simplicity the symetric case is considered

in (9). For it the error estimation is given as[2]:

$$error \le (b-a)\frac{\Delta x^2}{12}\max\left|\frac{d^2}{dx^2}e^{-ixp}f(x)\right| \tag{15}$$

$$= (b-a)\frac{\Delta x^2}{12}\max\left|-p^2e^{-ixp}f(x) - ipe^{-ixp}\frac{d}{dx}f(x) + e^{-ixp}\frac{d^2}{dx^2}f(x)\right| \tag{16}$$

$$\le (b-a)\frac{\Delta x^2}{12}\left[p_b^2\max|f(x)| + p_b\max\left|\frac{d}{dx}f(x)\right| + \max\left|\frac{d^2}{dx^2}f(x)\right|\right] \tag{17}$$

where p_b gives the region of transformed variable $-p_b < p < +p_b$. In the case where p_b is choosen to fully represent the transformed function, the function e^{-ip_bx} varies much faster than $f(x)$ therefore error can be estimated as:

$$error \le \frac{(b-a)\Delta x^2 p_b^2}{12}\max|f(x)| = \frac{(b-a)^3p_b^2}{12(N_0-1)^2}M \tag{18}$$

Calculating the reduced problem (6) the error becomes:

$$error\_reduced \le \frac{p_b^2}{12(N_0-1)^2} \tag{19}$$

which from previous equation is releated with error of general problem:

$$error = M(b-a)\cdot error\_reduced \tag{20}$$

3 Defining input and output and processing

3.1 Input

1. The function $f(x)$
2. Range of localisation $[a, b]$
3. Bound $|f(x)| \le M$
4. Range for output p_b
5. Spacing of output Δp
6. Absolute error *epsabs*

3.2 Output

1. Exact vlues for p_m
2. The calculated function values $\hat{f}(p_m)$

[2]Taken from Wikipedia *Trapezidoal Rule*

3.3 Algorithm

Firstly the problem is reformulated as reduced one (6):

$$epsabs \to \frac{epsabs}{M(b-a)} \tag{21}$$

$$\frac{f[(b-a)y+a]}{M} \to f(y) \tag{22}$$

$$(b-a)p \to p \tag{23}$$

where from last input aslo comes:

$$(b-a)p_b \to p_b \tag{24}$$

$$(b-a)\Delta p \to \Delta p \tag{25}$$

At the next step we are choosin the needed size of sample for integration or N_0 which according to (18) can be estimated as:

$$N_0 = \frac{p_b^2}{12 \cdot epsabs} \tag{26}$$

The all sample size N however is related to the resolution of output or Δp which from (10) gives us:

$$N \approx 2\pi \frac{N_0}{\Delta p} \tag{27}$$

At this point input vector is fully defined and can be fully evaluated with discrete Fourier transform or by repeating the (12), (10) one gets:

$$x_n = \Delta x n = \frac{n}{N_0 - 1} \tag{28}$$

$$p_m = p_m = 2\pi(N_0 - 1)\left(\frac{m}{N} - \frac{1}{2}\right) \tag{29}$$

$$\hat{f}(p_m) = \frac{1}{N_0 - 1} FFT\{(-1)^n \Theta(1 - x_n) f(x_n)\}_m \tag{30}$$

The last step is to use reduced problem results in order to come back to the general problem. From (6) it is:

$$(b-a)Me^{-iap_m/(b-a)}\hat{f}(p_m) \to \hat{f}_m \tag{31}$$

$$p_m/(b-a) \to p_m \tag{32}$$

4 Examples

After implementing the algorithm in *Python* I computed some specific examples which I compared with analitically known transforms[3].

[3] Taken from form section Fourier Transforms in Wikipedia

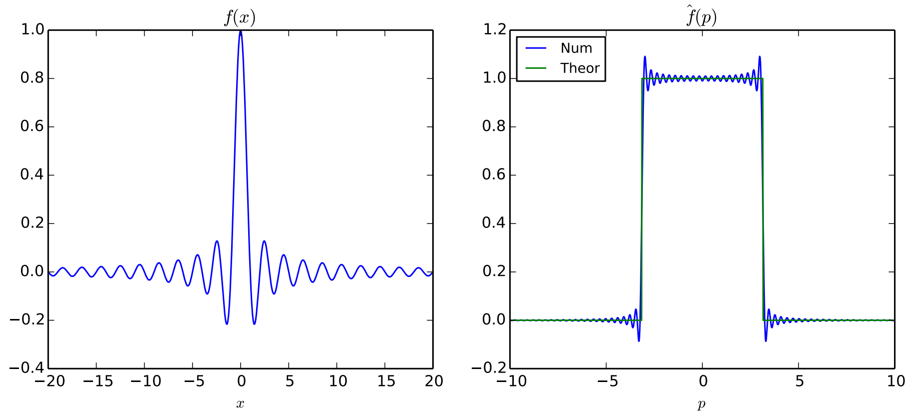

Figure 1: $sinc(x) \xrightarrow{F} rect(\frac{p}{2\pi})$

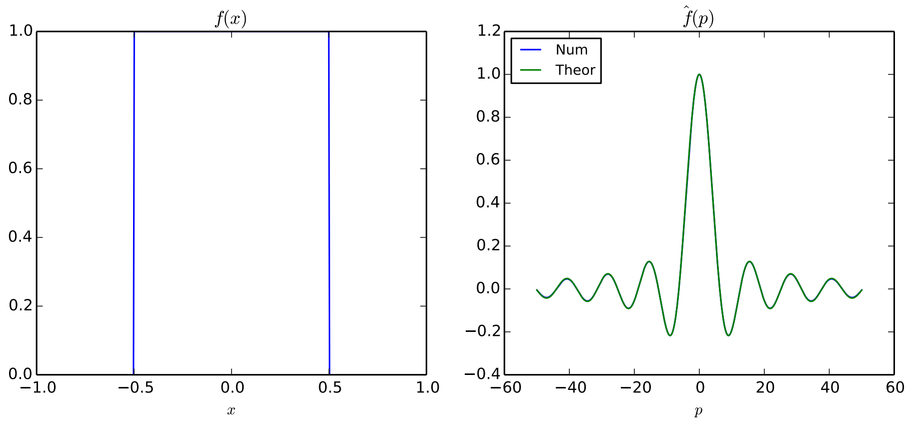

Figure 2: $rect(x) \xrightarrow{F} sinc(\frac{p}{2\pi})$

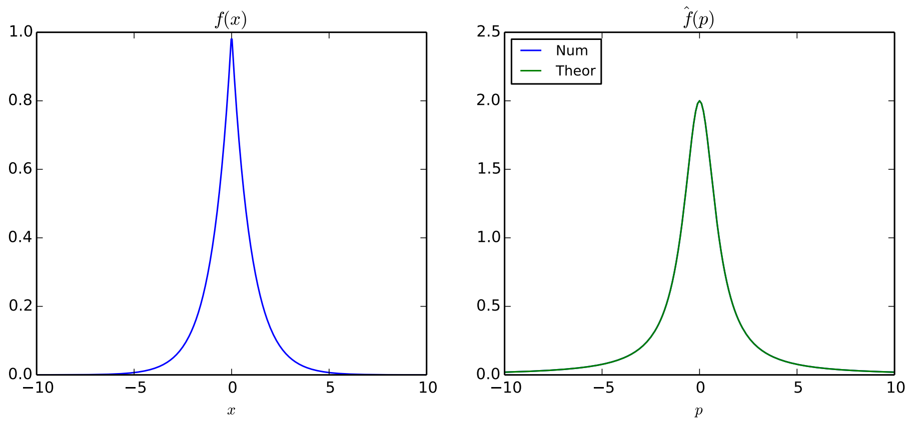

Figure 3: $e^{-|x|} \xrightarrow{F} \frac{2}{1+p^2}$

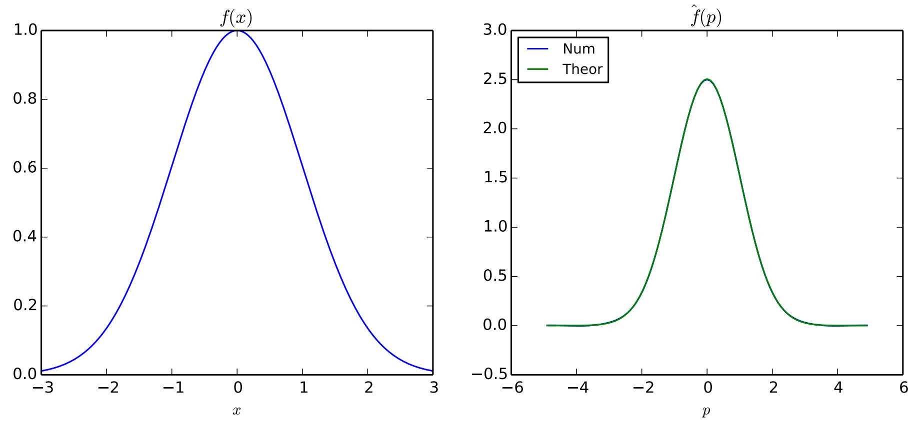

Figure 4: $e^{-x^2/2} \xrightarrow{F} \sqrt{2\pi} e^{-p^2/2}$

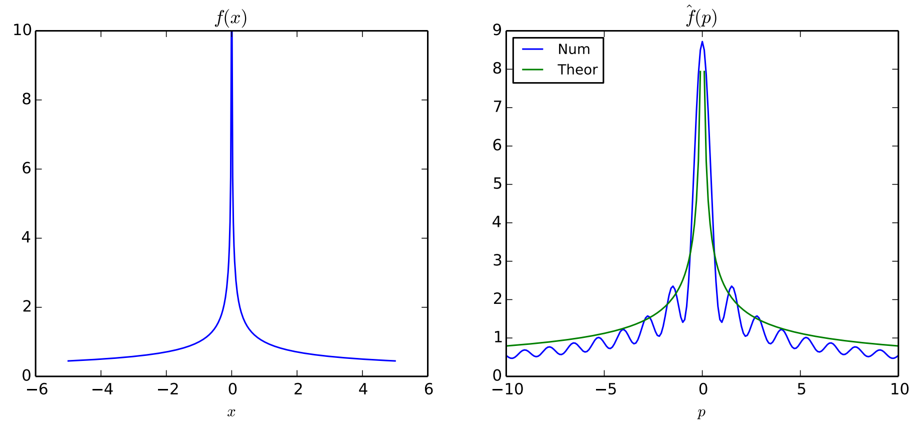

Figure 5: $\frac{1}{\sqrt{|x|}} \xrightarrow{F} \frac{\sqrt{2\pi}}{\sqrt{|p|}}$

5 Further development

- More examples of transformation is needed
- Performance should be checked
- Deeper error analysis should be performed numerically
- Range of localisation $[a, b]$ should be determined automatically.
- Bound M is possible to determine as Global minimum of $-|f(x)|$
- Range of output p_b could be estimated from normalisation condition or:

$$\int f(x)f^*(x)dx = \frac{1}{2\pi}\int \hat{f}(p)\hat{f}^*(p)dp \tag{33}$$

  with iterative process of enlarging sample size by factor of 2 (*even* and *odd* parts of FFT).

- Instead of explicitly giving p_b, Δp it could be given as linearly spaced vector (interpolation is unavoidable due to restriction of N, N_0 to be integers in (10))
- Due to [fourint] performance can be increased (as rule of thumb) by a facrotr of 3 on the Fourier transform.

6 Implementation in Python

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Jun  7 23:07:13 2014

@author: akels
"""

import numpy as np

def _fourier_N(f,N,N0):
    r"""
    Computes the:

    .. math::
        x_n &= \frac{n}{N_0 - 1} \\
        p_m &= 2 \pi (N_0 - 1) \left( \frac{m}{N} - \frac{1}{2} \right) \\
        n,m &= 0, 1,2 , \ldots, N-1 \\
        \hat{f}(p_m) &= \frac{1}{N_0 - 1} FFT \{(-1)^n \Theta(1-x_n) f(x_n) \}_m

    """
    n = np.arange(0,N0)
    x_n = n/(N0 - 1.)
    f_n = f(x_n)

    tilde_f = np.zeros(N)
    tilde_f[0:N0] = (-1)**n * f_n

    hat_f = np.fft.fft(tilde_f) / (N0 - 1.)

    return hat_f


def fourier_reduced(f,spacing,bound,epsabs=1e-4):
    r"""
    .. math::
```

```
                \hat{f}(p_m) &= \int_{0}^{1} e^{-i x p } f(x) dx \\
            p_m & \approx \Delta p m, ~ m=0,\pm 1, \pm 2, \ldots, p_b/\Delta_p \\
            |f(x)| &\le 1

    where :math:`\Delta p` is given as spacing but :math:`p_b` as bound.
    """

    N0 = int( bound/12 / np.sqrt(epsabs) ) # Without pedantic ceiling

    N = 2 * int( np.pi * N0/ spacing)

    print("N={}\t N0 = {}".format(N,N0))

    # For simplicity. Additional optimisation possible.
    m = np.arange(N)
    p_m = np.pi * (N0 - 1)*(2.*m/N - 1)
    valid = np.abs(p_m)< bound

    # Using already implemented
    hat_f_m = _fourier_N(f,N,N0)


    return p_m[valid], hat_f_m[valid]

def fourier(f,a,b,spacing,bound,M=1,epsabs=1e-2):
    r"""
    .. math::
        \hat{f}(p_m) &= \int_{a}^{b} e^{-i x p } f(x) dx \\
        p_m & \approx \Delta p m, ~ m=0,\pm 1, \pm 2, \ldots, p_b/\Delta_p \\
        |f(x)| &\le M

    where :math:`\Delta p` is given as spacing but :math:`p_b` as bound.
    """

    p_r,f_r = fourier_reduced(lambda y: f((b-a)*y + a)/M, spacing=(b-a)*spacing, bound = (b-a)*bound,epsabs = epsabs/M/(b-a))

    p = p_r/(b-a)

    f = (b-a)*np.exp(-1j*p*a)* M *f_r

    return p,f
```