Fourier Transform of Sinc Function

Contents

1 Problem Statement and Solution

问题起因在于计算积分式

$$\mathrm{pr}_h(\Delta|\bar{c}) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \cos \Delta t \prod_{m=k+1}^{k+h} \frac{\sin \bar{c}Q^m t}{\bar{c}Q^m t}\,\mathrm{d}t, \qquad k \in \mathbb{N}, \quad h \in \mathbb{N}_+, \quad \bar{c}, \Delta \in (0, \infty) \tag{1}$$

如果定义余弦 Fourier 变换为

$$\mathscr{F}_{\mathrm{c}}[f](\omega) \equiv \int_{-\infty}^{+\infty} f(t) \cos \omega t \,\mathrm{d}t \equiv \hat{f}(\omega) \tag{2}$$

那么 Eq (1) 变为

$$\mathrm{pr}_h(\Delta|\bar{c}) = \frac{1}{2\pi}\hat{f}(\Delta), \qquad f(t) \equiv \prod_{m=k+1}^{k+h} \frac{\sin \bar{c}Q^m t}{\bar{c}Q^m t} \tag{3}$$

下面讨论如何计算 sinc 连乘积的余弦 Fourier 变化。如果定义 $a_m \equiv \bar{c}Q^{k+1+m}, m = 0, 1, \dots, h-1$，那么

$$\begin{aligned}
\hat{f}(\Delta) &\equiv \int_{-\infty}^{+\infty} \cos \Delta t \prod_{m=0}^{h-1} \frac{\sin a_m t}{a_m t}\,\mathrm{d}t \\
&= \int_{-\infty}^{+\infty} \frac{\cos \Delta t}{(2\mathrm{i})^h} \prod_{m=0}^{h-1} \frac{e^{\mathrm{i}ta_m} - e^{-\mathrm{i}ta_m}}{a_m t}\,\mathrm{d}t \\
&= \int_{-\infty}^{+\infty} \frac{\cos \Delta t}{(2\mathrm{i}t)^h} \sum_{s\in\{-1,1\}^h} c_s e^{\mathrm{i}tb_s}\,\mathrm{d}t
\end{aligned} \tag{4}$$

最后一步计算连乘积时用到了排列数。这里 $s = (s_0, s_1, \dots, s_{h-1})$ 是分量取值为 ± 1 的 h 元数组。分量 $s_m = +1$ 对应连乘积第 m 项选用 $e^{\mathrm{i}ta_m}$，分量 $s_m = -1$ 对应连乘积第 m 项选用 $e^{-\mathrm{i}ta_m}$，这样容易算出

$$c_s = \prod_{m=0}^{h-1} \frac{s_m}{a_m}, \qquad b_s = \sum_{m=0}^{h-1} s_m a_m \tag{5}$$

考虑 $\{-1,1\}^h$ 的自映射 $\phi\colon s_m \mapsto -s_m$。$\phi$ 是双射，逆映射就等于自身，而且 $c_{\phi(s)} = (-1)^h c_s, b_{\phi(s)} = -b_s$,

$$\sum_{s\in\{-1,1\}^h} c_s e^{\mathrm{i}tb_s} = \sum_{\phi(s)\in\{-1,1\}^h} c_s e^{\mathrm{i}tb_s} = \sum_{s'\in\{-1,1\}^h} (-1)^h c_{s'} e^{-\mathrm{i}tb_{s'}} = \sum_{s\in\{-1,1\}^h} (-1)^h c_s e^{-\mathrm{i}tb_s} \tag{6}$$

这样 Eq (4) 就可以进一步化简，

$$\begin{aligned}\hat{f}(\Delta) &= \int_{-\infty}^{+\infty} \frac{1}{(2t)^h} \sum \frac{c_s e^{\mathrm{i}t(b_s+\Delta)} + c_s e^{\mathrm{i}t(b_s-\Delta)}}{2\mathrm{i}^h}\,\mathrm{d}t = \int_{-\infty}^{+\infty} \frac{1}{(2t)^h} \sum \frac{c_s e^{\mathrm{i}t(b_s+\Delta)} + (-1)^h c_s e^{\mathrm{i}t(-b_s-\Delta)}}{2e^{\mathrm{i}\pi h/2}}\,\mathrm{d}t \\ &= \int_{-\infty}^{+\infty} \frac{1}{(2t)^h} \sum c_s \cos\left[(b_s+\Delta)t - h\pi/2\right]\mathrm{d}t\end{aligned} \tag{7}$$

事实上，Eq (4)–(7) 就是完成了对被积函数 $f(t)\cos\Delta t$ 的恒等变形，

$$\begin{aligned}\cos\Delta t \prod_{m=0}^{h-1} \frac{\sin a_m t}{a_m t} &= \frac{1}{(2t)^h} \sum_{s\in\{-1,1\}^h} c_s \cos\left[(b_s+\Delta)t - h\pi/2\right] \\ 2^h \cos\Delta t \prod_{m=0}^{h-1} \sin a_m t/a_m &= \sum_{s\in\{-1,1\}^h} c_s \cos\left[(b_s+\Delta)t - h\pi/2\right] \equiv C_h(t)\end{aligned} \tag{8}$$

因为 $C_h(t)$ 含有 h 个 sin 因子，所以 $x=0$ 是其 h 阶零点，因此可以分部积分，

$$\begin{aligned}\hat{f}(\Delta) &= 2\int_0^{+\infty} \frac{C_h(t)}{(2t)^h}\,\mathrm{d}t = \frac{1}{2^{h-1}} \int_0^{+\infty} \frac{1}{h-1} \frac{C_h'(t)}{t^{h-1}}\,\mathrm{d}t \\ &= \frac{1}{2^{h-1}(h-1)!} \int_0^{+\infty} \frac{C_h^{(h-1)}(t)}{t}\,\mathrm{d}t = \frac{1}{2^{h-1}(h-1)!} \sum c_s (b_s+\Delta)^{h-1} \int_0^{+\infty} \frac{\sin(b_s+\Delta)t}{t}\,\mathrm{d}t \\ &= \frac{\pi}{2^h(h-1)!} \sum c_s (b_s+\Delta)^{h-1} \operatorname{sgn}(b_s+\Delta)\end{aligned} \tag{9}$$

这样就得出了积分 Eq (1) 的解析表达式，

$$\begin{aligned}\mathrm{pr}_h(\Delta|\vec{c}) &= \frac{1}{2^{h+1}(h-1)!} \sum_{s\in\{-1,1\}^h} c_s (b_s+\Delta)^{h-1} \operatorname{sgn}(b_s+\Delta) \\ c_s &= \prod_{m=0}^{h-1} \frac{s_m}{a_m}, \qquad b_s = \sum_{m=0}^{h-1} s_m a_m, \qquad a_m = \bar{c} Q^{k+1+m}\end{aligned} \tag{10}$$

2 Numeric Evaluation—Gaussian Quadrature

众所周知，在有界闭区间 $[a,b]$ 上的积分操作，记为 $\mathscr{I}$，是从一致连续函数空间 $C([a,b])$ 到实数域 $\mathbb{R}$ 的线性运算。解析上计算积分的一般方法是求原函数，辅助技巧包括分部积分、换元积分、逐项积分等等。缺点是技巧性太强，无法编程实现。数值上有一类机械求积公式 $\mathscr{Q}$，其本质也是一种线性变换。对于多项式函数，或者用多项式函数拟合得比较好的函数而言，两者相差不大，因此应用比较广泛。

$$\mathscr{Q}[f] \coloneqq \sum_{k=0}^{n-1} A_k f(x_k), \qquad \mathscr{R}[f] \coloneqq \mathscr{I}[f] - \mathscr{Q}[f], \tag{11}$$

$(A_k), (x_k)$ 分别称为求积结点和求积系数，而求积法则 $\mathscr{Q}$ 完全由此决定。$\mathscr{R}[f]$ 是积分余项，用于衡量误差大小。一般为了保证对于多项式类似函数的误差较小，会要求方程组

$$\mathscr{R}[x^m] = 0, \qquad m = 0, 1, 2, \dots, \tag{12}$$

尽可能多的成立。如果事先给定好 n 个彼此互异的求积结点 (x_k)，将求积系数 (A_k) 看作变量，那么一些简单的代数知识告诉我们，Eq (12) 中前 n 个方程同时成立时，系数矩阵行列式不为零，解是唯一存在的。如果要求结点 (x_k) 是在 $[a,b]$ 上均匀分布，联立 Eq (12) 前 n 个方程组并求解就可得出 (A_k)。这样给出的求积法则 Eq (11) 称作 Newton–Cotes 求积公式。因为直到 x^{n-1} 次方，Newton–Cotes 公式的误差都为零，即精确成立，所以称其具有至少 $n-1$ 次代数精度。但若将求积结点 (x_k) 也看作未知量，那么 Eq (12) 中前 $2n$ 个方程同时成立也是很有可能的。Gauss 求积公式就做到了这一点，通过选择结点 (x_k) 为 n 阶正交多项式 p_n 的零点，保证了其代数精度至少为 $2n-1$。在一般性地讨论正交多项式之前，先举一个最常用的例子，Legendre 多项式，作为引入。

$$\mathrm{P}_n = \frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\Big[(x^2-1)^n\Big], \qquad n = 0, 1, 2, \dots, \tag{13}$$

是一种比较方便的 Legendre 多项式的表示方法。容易看出 P_n 是最高次项为 $x^n/(2^n n!)$ 的多项式，而且具有明确的奇偶性。这里主要关注的是在区间 $[-1,1]$ 上的正交性。对于函数 $f, g \in C([-1,1])$，可以定义内积为 $(f,g) \coloneqq \mathscr{I}[fg]$，积分区间取为 $[-1,1]$。如果记 $\Phi(x)_n \coloneqq (x^2-1)^n/(2^n n!)$，那么有 $\mathscr{I}[\mathrm{P}_n\mathrm{P}_m] = \mathscr{I}[\Phi_n^{(n)}\Phi_m^{(m)}]$。因为 -1, 1 是 Φ_n 的两个 n 重根，所以 -1, 1 也是 $\Phi_n^{(n-k)}$ 的 k 重根。反复利用分部积分可知 (不妨设 $m \leqslant n$)

$$(\mathrm{P}_n, \mathrm{P}_m) = \mathscr{I}[\Phi_n^{(n)}\Phi_m^{(m)}] = \cdots = \mathscr{I}[(-1)^k\Phi_n^{(n-k)}\Phi_m^{(m+k)}] = \cdots = \mathscr{I}[(-1)^m\Phi_n^{(n-m)}\Phi_m^{(2m)}], \tag{14}$$

因为 Φ_m 是 $2m$ 次多项式，所以积分中后一项为常数。如果 $m < n$，那么 -1, 1 是 $\Phi_n^{(n-m)}$ 的原函数的 $m+1$ 重根，即积分等于零。如果 $m = n$，那么 Eq (14) 化为

$$\begin{aligned}(\mathrm{P}_n, \mathrm{P}_n) = \mathscr{I}[(-1)^n\Phi_n\Phi_n^{(2n)}] &= \int_{-1}^{1}\frac{(-1)^n(2n)!}{(2^n n!)^2}(x^2-1)^n\,\mathrm{d}x = \frac{(2n-1)!!}{(2n)!!}2\int_0^1(1-x^2)^n\,\mathrm{d}x \\ &= \frac{(2n-1)!!}{(2n)!!}2\int_0^{\pi/2}\cos^{2n+1}\theta\,\mathrm{d}\theta = \frac{(2n-1)!!}{(2n)!!}2\frac{(2n)!!}{(2n+1)!!} = \frac{2}{2n+1}\end{aligned}, \tag{15}$$

2.1 Orthogonal Polynomials

一般地，如果在 $\{1, x, x^2, \dots,\}$ 张成的无限维线性空间 $\boldsymbol{H}_\infty$ 上定义了内积，则称 $(p_k)_{k=0}^\infty$ 为正交多项式族，如果 p_k 是 k 次多项式，且彼此正交。需要注意到的是，一旦内积定义完成，那么正交多项式也就完全确定了 (在首项系数取为一的情形下)。因为 $p_0 \equiv 1$ 总是成立的，设 $p_1 = x + a_0$，那么只要根据正交关系 (p_1, p_0) 就可以求出待定系数，从而得到 p_1。一般来说，除去首项，n 次正交多项式 p_n 有 n 个待定系数而通过与 $p_0, p_1, \dots, p_{n-1}$ 正交关系可以完全确定这些系数，所以内积完全决定了正交多项式。从这个意义上说，如果定义 $\boldsymbol{H}_\infty$ 上的内积 (f,g) 为 fg 在区间 $[-1,1]$ 上的积分，那么得到的正交多项式就是 Legendre 多项式，满足表达式 Eq (13).

考虑 $\{1, x, x^2, \dots, x^m\}$ 张成的 $m+1$ 维线性空间 $\boldsymbol{H}_m$，从正交性容易看出 $\{p_0, p_1, \dots, p_m\}$ 线性无关，所以其是 $\boldsymbol{H}_m$ 的一组基，x^m 可由其线性表出。另一方面，如果 $n > m$，那么从正交性看出 p_n 和 $\boldsymbol{H}_m$ 的一组基正交，即 n 次正交多项式 p_n 和任何次数严格低于 n 的多项式正交。特别地，$(p_n, 1) = 0$。这说明 p_n 在被积区间上必然变号。设 p_n 的 n 个根为 (x_k)，那么可以断言 (x_k) 必然全部落在被积区间内。否则，如果只有 $m < n$ 个根 $(x_i)_{i\in I}$ 在被积区间内，那么 $p_n \cdot \prod_{i\in I}(x - x_i)$ 在被积区间内不变号，与 p_n 正交于 m 次多项式 $\prod_{i\in I}(x-x_i)$ 的结论矛盾。利用这种方法还可以证明所有的零点都是单根，而且不会出现在边界上。

2.2 General Gaussian Quadrature

一旦了解到这些关于正交多项式的简单性质，那么 n 个结点的 Gauss 求积公式 Eq (11) 能到达至少 $2n-1$ 的代数精度也就不难理解了。假设要求的是两个函数 f, ω 在闭区间 $[a,b]$ 上的积分，

$$\mathscr{I}[f] \coloneqq \int_a^b f(x)\cdot\omega(x)\,\mathrm{d}x, \tag{16}$$

引入函数 ω 的目的就是尽量保证 f 能用多项式函数很好的近似，毕竟我们知道，对很多初等函数，比如指数函数和三角函数，用多项式近似的效果并不好。然后定义 $\boldsymbol{H}_\infty$ 上的内积为 $(f,g)=\mathscr{I}[fg]$。一般我们要求 $\omega\in C([a,b])$，且在被积区间上不变号，因为这样可以保证 $\mathscr{I}[fg]$ 确实是一个内积。而根据之前的介绍，内积可以引出正交多项式族 (p_n)。取 n 个求积结点 (x_k) 为 n 次正交多项式的 p_n 零点 (从前面分析知，p_n 的零点确实尽数落在 $[a,b]$ 上)。取 n 个求积系数 (A_k) 为 Eq (12) 中前 n 个方程联立的解 (解总是存在的)。那么求积公式 $\mathscr{Q}$ 至少具有 $n-1$ 次代数精度，即对 $\boldsymbol{H}_{n-1}$ 中的所有多项式都精确成立。更近一步，因为求积结点是 p_n 的根，所以 $\mathscr{Q}[x^m p_n]$ 总等于零。如果 $m<n$，那么从正交性看出 $\mathscr{I}[x^m p_n]$ 也等于零，说明对 $x^m p_n$ 形式的多项式，Gauss 求积公式 $\mathscr{Q}$ 总是精确成立的。一点简单的代数知识告诉我们，任给 $2n-1$ 次多项式 $f\in\boldsymbol{H}_{2n-1}$，可以用 p_n 作多项式除法

$$f = hp_n + r, \qquad h,r\in\boldsymbol{H}_{n-1}, \tag{17}$$

因为对 hp_n 和 r 而言，$\mathscr{Q}$ 都是精确成立的，所以对 $\boldsymbol{H}_{2n-1}$ 中任何多项式 f 而言，$\mathscr{Q}$ 都是精确成立的。自然就有 Gauss 求积公式具有至少 $2n-1$ 次代数精度。根据区间 $[a,b]$ 和 ω (称为权函数) 的不同，Gauss 积分有许多变种 (Wiki: Gaussian Quadrature)，使用时一般要预先利用换元等技巧将待求积分写成 Eq (16) 的形式。然后查表确定正交多项式种类，再确定 $(A_k),(x_k)$。需要说明的是，真正计算求积系数和求积结点的实用算法比较复杂，和这里给出的构造方法有很大区别。一种办法是根据正交多项式的三项递推关系，将求根问题转化成求某个三对角矩阵特征值，然后用 QR 分解的办法计算[1]。

Interval	weight function	Orthogonal Polynomials	Name
$[-1,1]$	1	Legendre Polynomials	Gauss-Legendre Qaudrature
$(-1,1)$	$(1-x)^\alpha(1+x)^\beta,\quad \alpha,\beta>-1$	Jacobi Polynomials	Gauss-Jacobi Qaudrature
$[0,\infty)$	$x^\alpha e^{-x},\quad \alpha>-1$	Laguerre Polynomials	Gauss-Laguerre Qaudrature
$(-\infty,\infty)$	e^{-x^2}	Hermite Polynomials	Gauss-Hermite Qaudrature

2.3 Adaptive Method

积分计算的一大难点在于被积函数的形式千变万化，难以用一个统一的形式去描述。Gauss 求积公式 Eq (11) 可以理解为用有限个特定的结点 (x_k) 处的函数值来刻画函数。当然，这种简化办法不能应对所有的情形。毕竟，被积函数的定义域是一整段区间，有限个点处的取值并不能完全代表一个函数。一种办法是不断增加结点的数量，让考察的函数值足够多。理论上可以证明，在 $n\to\infty$ 时，Gauss 求积公式给出的结果收敛于待求积分值。但困难在于，求积系数 (A_k) 和结点 (x_k) 的计算也不简单，一般都是预先算好后直接作为常数写入程序的。而没有一种好的办法做到将无穷多种可能用到的 $(A_k),(x_k)$ 全部输入到程序中。通过提高代数精度的办法得到的序列收敛速度也没有保证。

另一种办法是将分割被积区间，用数个子区间上的积分和来近似原区间上的积分。但如何分割区间也是一个问题。可以人为给定分割办法，比如均分为指定个等长的子区间。但更好的办法是让

[1] Gene H. Golub and John H. Welsch. *Calculation of Gauss Quadrature Rules*. Tech. rep. Stanford, CA, USA, 1967.

程序借助某个标准自行判断。假设 `Q(f, a, b)` 是一个数值积分的程序，返回两个数值 `i, e`，前者是 f 在区间 $[a,b]$ 上的积分估计值，而后者是误差大小的估计。这种估计不一定要十分精确，只需要大致保证 `e` 和实际误差正相关就好。比如，假设 $\mathcal{Q}_n^{\mathrm{GL}}, \mathcal{Q}_{n'}^{\mathrm{GL}}$ 是两个不同阶数的 Gauss-Legendre 求积公式，那么可以令 `i` 等于高阶公式给出的结果，而令 `e` 等于高阶与低阶结果之差的绝对值。这样，当计算在某个子区间 $[a_i,b_i]$ 上的积分时，可以通过返回值 `e` 的大小为标准来判断是否需要继续细分。这样的积分算法称为 (Local) Adaptive Quadrature。用 Python 代码可以写为，

Local Adaptive Quadrature

```
1  def QA(f, a, b, eps):
2      i, e = Q(f, a, b)
3      if e < eps:
4          return i
5      else:
6          m = (a+b)/2
7          return QA(f, a, m, eps/2) + QA(f, m, b, eps/2)
```

和基本的积分算法 `Q` 相比，适应型积分 `QA` 多出一个控制参数 `eps`。如果估计误差 `e` 小于 `eps`，则返回当前积分，否则，对半分割区间后递归计算。称其为局域适应算法的原因在于 Line 7，当被积区间减半后，允许误差 `eps` 也要减半，即相当于要求误差与区间长度的比值要小于某个给定值。对有些函数而言，这种标准有时可能难以达到。经常碰见的情形是，函数在定义域的某一小段区间上行为不好，估计误差的下降不如区间长度减半的速度，导致程序不停地尝试细分这段区间，消耗大量计算资源甚至直接崩溃。相比来说，有时全局的适应算法可以避免这种情况的出现。全局算法的思想是控制划分区间的顺序，优先对估计误差最大的区间进行细分。这样做的好处在于容易控制计算深度，比如最多只划分 50 次后就停止计算，直接返回所有子区间的积分之和。

Global Adaptive Quadrature

```
1  def QA(f, a, b, eps):
2      astack = EmptyStack()
3      max_depth = 50
4
5      # initialize astack with [a,b]
6      itv = [a,b]
7      itg, err = Q(f, a, b)
8      astack.push(Triple(itv, itg, err))
9
10     # main lopp
11     while astack.length <= max_depth:
12         if astack.sum_err() < eps:        # exit condition
13             return astack.sum_itg()
14
15         itv, itg, err = astack.get_triple_with_largest_err()
16
17         # refine interval
18         _a, _b = itv
19         _m = (_a + _b)/2
20         itg1, err1 = Q(f, _a, _m)
21         itg2, err2 = Q(f, _m, _b)
22
23         # update stack
```

```
24          astack.remove(Triple(itv, itg, err))
25          astack.push(Triple([_a,_m], itg1, err1))
26          astack.push(Triple([_m,_b], itg2, err2))
27
28      return -1                                   # won't happen usually
```

首先，为了方便控制递归的深度，以及选择如何划分区间，可以通过显示地构造一个『栈』型数据结构 `astack` 来存储计算数据，将程序转换成非递归的结构 (Line 2)。栈 `astack` 中存储的数据是三元数组 `triple`，包含的数据分别为：子区间 `itv`，积分估计值 `itg` 和误差估计值 `err` (Line 8)。当栈显示误差之和小于给定 (全局) 误差上界 `eps` 时，停止计算，返回栈中积分之和 (Line 12, 13)。否则，找到误差估计最大的子区间，并对其细分 (Line 18–24)。

无论是局域适应算法还是全局适应算法，实质上都只是对基本积分算法 `Q` 的封装，并不涉及实际的积分计算。但 `Q(f, a, b)` 具体要如何实现呢？到底如何去估计或者说衡量误差大小？在前面已经提到，可以用两个不同阶的 Gauss 求积公式，比如 n 结点和 $n+1$ 结点的公式，结果之差的绝对值 $|\mathcal{Q}_n^{\mathrm{GL}} - \mathcal{Q}_{n+1}^{\mathrm{GL}}|$ 作为误差估计值。只是，一般来说，正交多项式 p_n, p_{n+1} 的零点并不重合。也就是说，这样的效果相当于总共考虑 $n+(n+1)$ 个点处的函数值，而实际代数精度却只有 $2n+1$ (因为返回值为 $\mathcal{Q}_{n+1}^{\mathrm{GL}}[f]$)。为了有效利用结点处函数值，Kronrod 引入了另一种积分公式，

$$\mathcal{Q}[f] = \sum_{\nu=0}^{n-1} \sigma_\nu f(x_\nu) + \sum_{\mu=0}^{n} \sigma_\mu^* f(x_\mu^*), \tag{18}$$

选择 (x_ν) 为区间 $[a,b]$ 上关于权函数 ω 的 n 次正交多项式 p_n 的零点，选择 (x_μ^*) 为 $n+1$ 次 Stieltjes 多项式 E_{n+1} 的零点，并令 $2n+1$ 个权系数 $(\sigma_\nu), (\sigma_\mu^*)$ 为 Eq (12) 中前 $2n+1$ 个方程的解，那么遵循前面推导 Gauss 求积公式 Eq (11) 的方法，可以证明 Eq (18) 的代数精度可以达到 $3n+1$。关键利用在于 Stieltjes 多项式 E_{n+1} 的如下性质 (或者说定义)[2]

$$\mathcal{J}[x^m p_n E_{n+1}] = (x^m, p_n E_{n+1}) = 0, \qquad m = 0, 1, \ldots, n, \tag{19}$$

如果将求积公式 Eq (18) 记为 $\mathcal{Q}_{2n+1}^{\mathrm{GK}}$，那么以 $|\mathcal{Q}_{2n+1}^{\mathrm{GK}} - \mathcal{Q}_n^{\mathrm{GL}}|$ 做误差估计值，以 $\mathcal{Q}_{2n+1}^{\mathrm{GK}}$ 做积分估计值，最终效果是用 $2n+1$ 个点处的函数值达到了 $3n+1$ 的代数精度。这种积分方法称为 Gauss-Kronrod Quadrature。更多估计误差的方法可以参考相关文献[3]。

2.4 Implementation and Test

这里实现了两种数值积分算法。一种是非适应算法 `QNGL`，将区间 `itv` 等分为 `dvs` 份后，在每个小区间上应用 30 点 Gauss-Legendre 求积公式 Eq (11)。一种是适应算法 `QAGK`，以 15 点 Gauss-Kronrod 求积公式 Eq (18) 及相应的误差估计方法为基础积分算法，应用 Global Adaptive Quadrature 的框架计算积分，可接受控制参数 `eps, lmt`，分别为最大允许误差和最大允许迭代次数。求积系数和求积结点数据来自网站 ADVANPIX。测试结果格式如下，

Output Format

```
integrand: xto29
 interval: [0, 1]
```

[2] Franz Peherstorfer. "Stieltjes polynomials and functions of the second kind". In: 65.1 (1995), pp. 319–338.

[3] Pedro Gonnet. *A Review of Error Estimation in Adaptive Quadrature*. 2010.

```
3      quad rule: mquad
4   n evaluation:       189     (total of 3 times)
5  time per call: 0.00003982 sec (best of 3 times)
6  integral, err: 0.03333333, 0.00000000
7            ref: 0.03333333, 0.00000000
8       add info: None
```

这里主要考察的是运行时间 (Line 5) 以及计算函数在某点处值的次数 (Line 4)。Line 7, 8 分别是算法给出的积分估计值，误差值，和理论给出的积分值，估计值和理论值的绝对误差。`QNGL` 和 `QAGK` 的测试结果表明，无论是指数函数 $e^{-x},(-\infty,0)$，还是三角函数 $\sin x,[0,\pi/2]$，以及多项式函数 $x^{29},[0,1]$，理论值与实际值的误差都在 10^{-9} 以内。

最后是对积分式 Eq (1) 的计算。为了更有参考价值，除去 `QNGL`, `QAGK`，这里还实现了另外两种计算积分的办法。一种方法基于前面解析得出的 Eq (10)。为了避免舍入误差过大，需要对计算方法做一些微调，

$$\mathrm{pr}_h(\Delta|\tilde{c}) = \frac{\sum c_s (b_s+d)^{h-1}\,\mathrm{sgn}\,(b_s+d)}{2^{h+1}(h-1)!\tilde{c}Q^{k+1}\prod a_m}, \qquad d=\frac{\Delta}{\tilde{c}Q^{k+1}}\in(0,+\infty), \qquad s=(s_0,s_1,\dots,s_{h-1})\in\{-1,1\}^h,$$

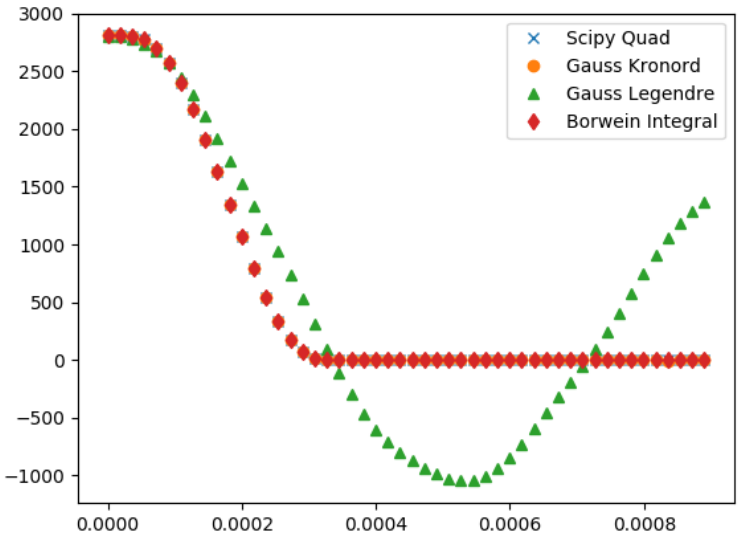
$$a_m = Q^{m-1}, \qquad b_s=\sum_{m=0}^{h-1} s_m a_m, \qquad c_s=\prod_{m=0}^{h-1} s_m. \tag{20}$$

比较关键的一点在于通过换元积分将 $\Delta,\tilde{c}$ 整合为一个常数 d。根据 Fourier 变化的一些知识，可以断言 $d>\sum Q^m$ 时，积分是等于零的[4]。大致来说可以这样理解，因为整个积分可以看作 $\prod \mathrm{sinc}\,\tilde{c}Q^{k+1+m}t$ 的 FCT 在 Δ 处的函数值。而根据卷积定理，这等于 h 个 $\mathrm{sinc}\,\tilde{c}Q^{k+1+mt}$ 的 FCT 的卷积在 Δ 处的值。因为 $\mathrm{sinc}\,\tilde{c}Q^{k+1+m}t$ 的 FCT 在区间 $[-\tilde{c}Q^{k+1+m},\tilde{c}Q^{k+1+m}]$ 之外恒等于零。而容易看出，如果 f,g 分别只在 $[-a,a],[-b,b]$ 上不为零，那么 f 与 g 的卷积必然只在 $[-a-b,a+b]$ 上不为零。所以用数学归纳法可以看出当 $\Delta>\sum\tilde{c}Q^{k+1+m}$ 时，积分必然为零。

另一种办法是利用 Python 的科学计算库 SciPy 提供的数值积分方法 `nquad`，其内部调用的是 FORTRAN QUADPACK 库 (文档链接)，实现方法类似全局适应型 Gauss-Kronrod 求积公式，但使用了特殊的加速方法。

另外，前面实现的 `QNGL`, `QAGK` 都只能处理有界区间的情形，直接计算 Eq (1) 之前要先进行换元，将无界区间映射到有界区间。这里采用的办法是先利用对称性将积分区间变为 $[0,\infty)$，然后使用变换 $t=1-1/x$，将被积区间转换为 $[0,1]$。注意，因为原被积函数在无穷远处极限存在，所以换元后的积分区间可以是闭的，即端点不是奇点 (否则应当使用 Jacobi 正交多项式)。

[4] David Borwein and Jonathan M. Borwein. "Some Remarkable Properties of Sinc and Related Integrals". In: ***The Ramanujan Journal*** 5.1 (Mar. 2001), pp. 73–89.



上图是固定 $\bar{c}, Q, k, h$ 分别为 $.01, .511, 5, 10$ 后，积分值 $\mathrm{pr}_h(\Delta|\bar{c})$ 在区间 $[0, .05Q^{k+1}]$ 上的函数图像。可以看出非适应的 Gauss-Legendre 求积公式在 Δ 变大后越来越不准确，而两种适应算法和解析公式的结果符合的很好，三条线几乎完美重合，其中又以 `QAGK` 和 `nquad` 最为接近，是解析公式 Eq (20) 到后者的距离的 1/3 左右。

实现代码见链接 Gaussian-Quadrature. `gaussquad.py` 是实现了积分算法 `QNGL`, `QAGK`, `driver.py` 用于测试和分析算法速度和误差，`practice.py` 用于比较四种计算目标积分的方法，并完成绘图。