4-18-2019 11:00 AM

A Survey Of Numerical Quadrature Methods For Highly Oscillatory Integrals

Jeet Trivedi, *The University of Western Ontario*

Supervisor: Corless, Robert M., *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Applied Mathematics
© Jeet Trivedi 2019

Abstract

In this thesis, we examine the main types of numerical quadrature methods for a special subclass of one-dimensional highly oscillatory integrals. Along with a presentation of the methods themselves and the error bounds, the thesis contains implementations of the methods in Maple and Python. The implementations take advantage of the symbolic computational abilities of Maple and allow for a larger class of problems to be solved with greater ease to the user. We also present a new variation on Levin integration which uses differentiation matrices in various interpolation bases.

Keywords: Highly Oscillatory Quadrature, Numerical Quadrature, Levin Type Methods, Filon Integration, Asymptotic Methods, Moment Free integration

Contents

List of Figures

Chapter 1

Introduction

In this thesis, we will primarily discuss and present methods that numerically evaluate highly oscillatory integrals of the form

$$\int_a^b f(x)e^{i\omega g(x)}\,dx,$$

where $f(x)$ and $g(x)$ are sufficiently smooth real valued functions and $\omega \in \mathbb{R}$ such that $|\omega| \gg 1$.

Integrals of this form arise in a broad range of applications such as radiative transfer [7], fluid dynamics and electrodynamics [15]. They are also a extensively studied in harmonic analysis and computational harmonic analysis [22].

1.1 Failure of Classical Quadrature

Consider the following example from [10]:

$$\int_{-1}^{1} \cos(x)e^{i\omega x^2}\,dx. \tag{1.1}$$

A plot of the integrand for $\omega = 50$ (Figure (1.1)) reveals its highly oscillatory nature.

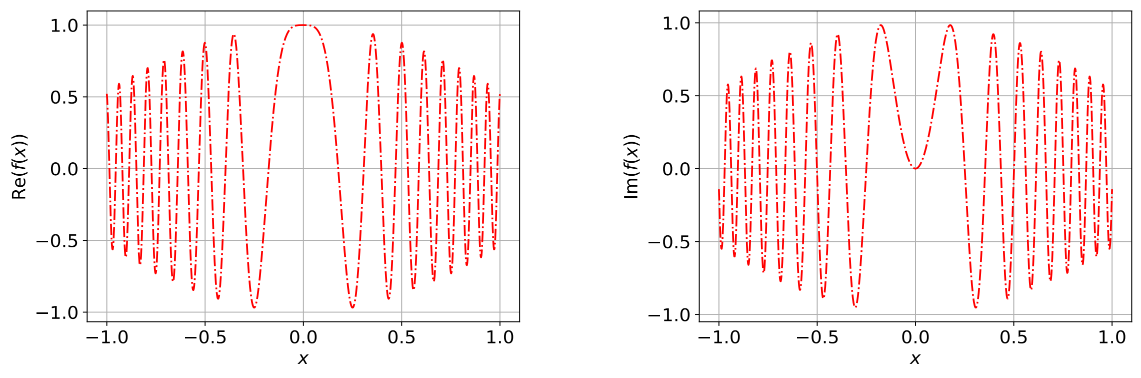


Figure 1.1: Plot of the real and complex parts of the integrand $f(x) = \cos(x)e^{i\omega x^2}$ for $\omega = 50$.

Typical naive numerical quadrature methods rely on sampling points and interpolating the integrand over a basis of functions for which the integrals over the intervals are precomputed. From the plots in Figure (1.1), it is clear that we will need to sample a large number of points in order to get a reasonable numerical approximation for this function. Furthermore, the number of points that need to be sampled will grow as we increase ω.

Let's attempt to use Gaussian quadrature to evaluate integral (1.1). With 10 points sampled,

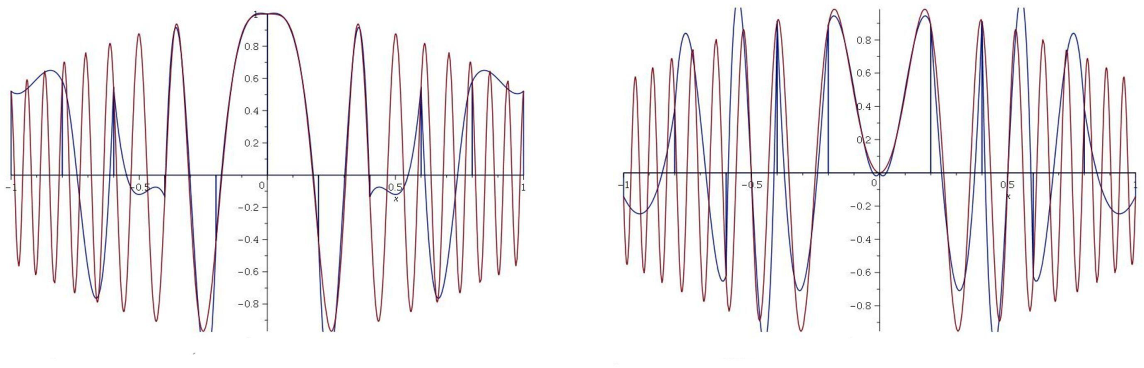


Figure 1.2: Plots of the interpolated curves used for Gaussian quadrature. The red indicates the function while the blue indicates the interpolated curves used for Gaussian quadrature.

From plots (1.2), it is clear that standard quadrature is not enough. More so, if we fix the number of partitions and raise ω, we see in plot (1.3) that the error is asymptotically $O(1)$.

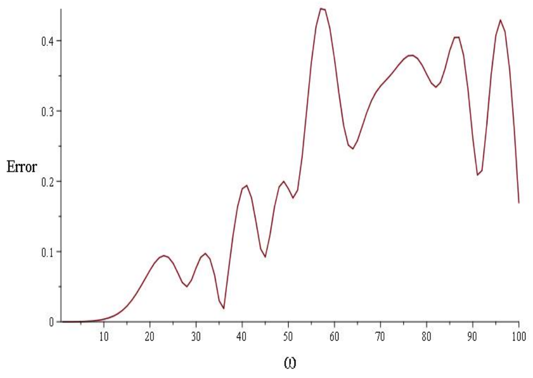


Figure 1.3: Error in Gaussian quadrature of integral (1.1) as a function of ω.

The method we will examine will allow us to compute integrals of this type much more efficiently with an remarkable property. The absolute error will actually decrease as ω increases.

The road map for the rest of chapters is as follows: First we examine two classical methods, the asymptotic method and the Filon method. During the examination, we will naturally find the need to define the methods differently for the case when $g(x)$ has stationary points in the domain and when it doesn't. With the classical methods presented, we will then look at the Levin method, which is a moment-free method but cannot handle integrals with stationary points. Then we will examine moment-free variants of both the Asymptotic and the Filon method in the presence of stationary points. The moment-free versions, while essentially solving the problem, require that $g''(x) \neq 0$ in the integration domain (except the stationary point, of course). This leads us then to the final chapter, which combines all these methods and provides a higher level interface to which the user can simply pass the integral and the integration domain is divided appropriately and the correct method used. This "hybrid" method is one that can be easily integrated[1] into any existing computer algebra system's integration library.

Lastly, the Appendix contains the code for all these methods in Maple, which in itself is a major part of the thesis. The same code is available for download through the Github (`https://github.com/jeettrivedi/highly-oscillatory/`).

[1]no pun intended

Chapter 2

Asymptotic Type Methods

Here we present the treatment of the material from [13]. As the name might suggest, this method relies on us being able to asymptotically expand the integral

$$\int_a^b f(x)e^{i\omega g(x)}\,dx. \tag{2.1}$$

We will divide our presentation into two parts, first only considering monotone $g(x)$ (equivalently, $g(x)$ has no stationary point in $[a,b]$) and then the general case. The reason for this separation in cases will become clear when we derive the asymptotic expansion. Let us start by assuming that $g(x)$ has no stationary points in $[a,b]$.

2.1 In The Absence of Stationary Points

In order to present the derivation cleanly, make a few definitions. First, let

$$I[f] := \int_a^b f(x)e^{i\omega g(x)}\,dx.$$

Next, consider the integral,

$$I[f\cdot(g')^2] = \int_a^b f(x)\cdot(g'(x))^2 e^{i\omega g(x)}\,dx \tag{2.2}$$

Using integration by parts we get

$$\begin{aligned} I[f\cdot(g')^2] =& \frac{1}{i\omega}\left[f(x)g'(x)e^{i\omega g(x)}\right]\Big|_a^b - \frac{1}{i\omega}\int_a^b \frac{d}{dx}(f(x)g'(x))e^{i\omega g(x)}\,dx \\ =& \frac{1}{i\omega}\left[f(x)g'(x)e^{i\omega g(x)}\right]\Big|_a^b - \frac{1}{i\omega}I\left[\frac{d}{dx}(fg')\right] \end{aligned} \tag{2.3}$$

Now, because we assumed that $g(x)$ has no stationary points in $[a,b]$, we can replace $f(x)$ by $\frac{f(x)}{g'(x)^2}$ in the expansion (2.3), which results in

$$I\left[\frac{f}{(g')^2}\cdot(g')^2\right] = I[f] = \frac{1}{i\omega}\left[\frac{f(x)}{g'(x)}e^{i\omega g(x)}\right]\Big|_a^b - \frac{1}{i\omega}I\left[\frac{d}{dx}\left(\frac{f}{g'}\right)\right]. \tag{2.4}$$

Let us now expand the last term of expansion (2.4) strategically using integration by parts.

$$I\left[\frac{d}{dx}\left(\frac{f}{g'}\right)\right] = \left(\int g'(x)e^{i\omega g(x)}\right)\left[\frac{1}{g'(x)}\left(\frac{d}{dx}\frac{f}{g'}\right)\right]_a^b - \int_a^b \frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)\left(\int g'(u)e^{i\omega g(u)}\,du\right)dx \tag{2.5}$$

Now noticing that $\int^x i\omega g'(u)e^{i\omega g(u)}\,du = e^{i\omega g(u)}$, we get

$$I\left[\frac{d}{dx}\left(\frac{f}{g'}\right)\right] = \frac{1}{i\omega}\left[\frac{e^{i\omega g(x)}}{g'(x)}\frac{d}{dx}\left(\frac{f}{g'}\right)\right]_a^b - \frac{1}{i\omega}\int_a^b \frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)e^{i\omega g(x)}\,dx, \tag{2.6}$$

$$I\left[\frac{d}{dx}\left(\frac{f}{g'}\right)\right] = \frac{1}{i\omega}\left[\frac{e^{i\omega g(x)}}{g'(x)}\frac{d}{dx}\left(\frac{f}{g'}\right)\right]_a^b - \frac{1}{i\omega}I\left[\frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)\right]. \tag{2.7}$$

And so, collecting everything, we have

$$I[f] = \frac{1}{i\omega}\left[\frac{f(x)}{g'(x)}e^{i\omega g(x)}\right]_a^b - \frac{1}{(i\omega)^2}\left[\frac{e^{i\omega g(x)}}{g'(x)}\frac{d}{dx}\left(\frac{f}{g'}\right)\right]_a^b + \frac{1}{(i\omega)^2}I\left[\frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)\right]. \tag{2.8}$$

Another application of integration by parts yields,

$$\begin{aligned} I[f] = {} & \frac{1}{i\omega}\left[\frac{f(x)}{g'(x)}e^{i\omega g(x)}\right]_a^b - \frac{1}{(i\omega)^2}\left[\frac{e^{i\omega g(x)}}{g'(x)}\frac{d}{dx}\left(\frac{f}{g'}\right)\right]_a^b + \frac{1}{(i\omega)^3}\left[\frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)\frac{e^{i\omega g(x)}}{g'(x)}\right]_a^b \\ & - \frac{1}{(i\omega)^3}I\left[\frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{1}{g'}\frac{d}{dx}\left(\frac{f}{g'}\right)\right)\right)\right] \end{aligned} \tag{2.9}$$

From the expression (2.9), a pattern becomes apparent. In each successive term, there is an additional $\frac{1}{i\omega g'(x)}\frac{d}{dx}$ of the non-exponential part of the previous term. We can formalize this pattern as follows. Let

$$f_0(x) = f(x) \tag{2.10}$$

$$\text{and } f_{m+1}(x) = \frac{d}{dx}\left(\frac{f_m(x)}{g'(x)}\right) \tag{2.11}$$

Then the asymptotic expansion is

$$I[f] \sim \sum_{m=0}^{\infty} -\frac{1}{(i\omega)^{m+1}}\left[\frac{e^{i\omega g(x)}}{g'(x)}f_m(x)\right]_a^b = -\sum_{m=0}^{\infty}\frac{1}{(i\omega)^{m+1}}\left[\frac{e^{i\omega g(b)}}{g'(b)}f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)}f_m(a)\right]. \tag{2.12}$$

Therefore an asymptotic method with p terms (denoted Q_p^A) is simply this series terminated after p terms. That is

$$Q_p^A[f,g] \equiv -\sum_{m=0}^{p-1}\frac{1}{(-i\omega)^{m+1}}\left[\frac{e^{i\omega g(b)}}{g'(b)}f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)}f_m(a)\right]. \tag{2.13}$$

An immediate consequence of truncating the series to the first p terms is that the asymptotic error in ω is

$$Q_p^A[f,g] - I[f] \sim O(\omega^{-p-1}) \tag{2.14}$$

Now with the approximation at hand, let us quickly gather our thoughts. Firstly, the derivation makes clear why special attention is required in the presence of stationary points as the substitution we make of $\frac{f}{(g')^2}$ can't be done in the presence of stationary points. Secondly, equation (2.14) makes it clear why the error decreases as ω increases.

Lastly, as we only need to take symbolic derivatives, we can use computer algebra systems such as Maple to automate this process of computing the series and automate the entire method. We note in passing that this theme of automating these methods using computer algebra systems will prevail throughout this discussion.

2.1.1 Numerical Examples

Example 2.1.1 $\int_a^b f(x)e^{i\omega x}\,dx$

In this particular case, the sequence of functions $f_m(x)$ has a simple form because $g'(x) = 1$. So, we have the following form,

$$Q_p^A[f,x] = -\sum_{m=0}^{p-1} \frac{1}{(-i\omega)^{m+1}} \left[e^{i\omega b} f^{(m)}(b) - e^{i\omega a} f^{(m)}(a)\right]. \tag{2.15}$$

Furthermore, if $g(x)$ is monotone in $[a,b]$ then we know the function $g(x)$ is invertible on $[a,b]$. So, we can use the expansion (2.15) to evaluate any integral of the form (2.1) as with a simple change of variables. We get

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = \int_{g(a)}^{g(b)} \frac{f(g^{-1}(x))}{g'(g^{-1}(x))} e^{i\omega x}\,dx.$$

Now we demonstrate with some examples that the error is indeed $O(\omega^{-p-1})$. The code used to calculate the examples in this section is included in (A.1.1).

Example 2.1.2 $I = \int_{-1}^1 \cos(x)e^{i\omega x}\,dx$

The integral has the closed form expression

$$\int_{-1}^1 \cos(x)e^{i\omega x}\,dx = \frac{-e^{-i\omega}}{\omega^2 - 1}(i\omega\cos(1)e^{2i\omega} + \sin(1)e^{2i\omega} - i\omega\cos(1) + \sin(1)). \tag{2.16}$$

Recall that we expect the error to be $O(\omega^{-p-1})$. This however, makes no promises about the magnitude of the error but only about the decay as $\omega \to \infty$. Let's calculate the expression $Q_2^A[\cos(x), x]$.

$$Q_2^A[\cos(x),x] = -\frac{1}{(-i\omega)}(e^{i\omega}\cos(1) - e^{-i\omega}\cos(1)) + \frac{1}{(-i\omega)^2}(e^{i\omega}\sin(1) - e^{-i\omega}\sin(1)) \tag{2.17}$$

Let's plot the actual error $||Q_2^A - I||$ on a logarithmic scale and the error scaled by ω^3, that is $||Q_2^A - I|| \cdot \omega^3$.

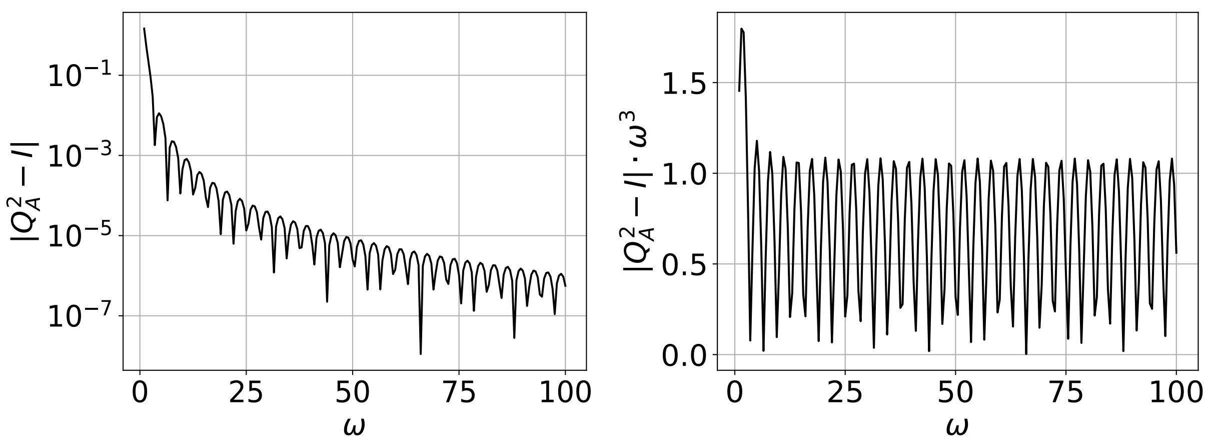


Figure 2.1: *Left:* Error plotted on a logarithmic scale of using the expression (2.17) to evaluate $\int_{-1}^{1} \cos(x) e^{i\omega x}\, dx$. *Right:* Absolute Error of using the expression (2.17) to evaluate $\int_{-1}^{1} \cos(x) e^{i\omega x}\, dx$ scaled by ω^3.

*The first thing to notice from the log plot (**??**) is that the absolute error is on the scale of* 10^{-5} *and decreases as ω increases. Furthermore, from the scaled error plot (**??**) we see that it is indeed decaying as* ω^{-3}.

But perhaps this integral was too simple. Let's attempt to evaluate an integral which does not have a closed form solution in terms of elementary functions.

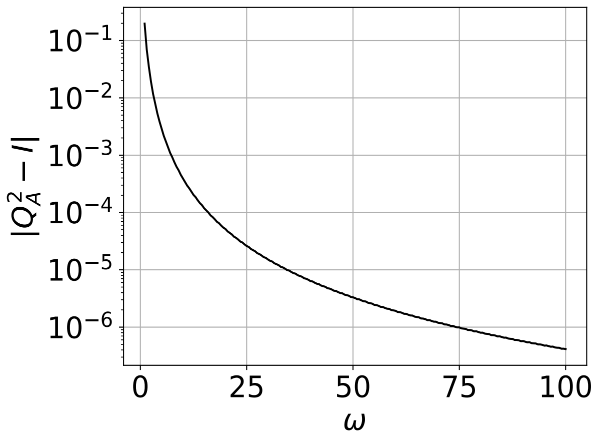
Example 2.1.3 $\int_{-1}^{1} \sin(x) e^{i\omega(x-2)^2}\, dx$

Using Maple, we obtain a closed form of the integral.

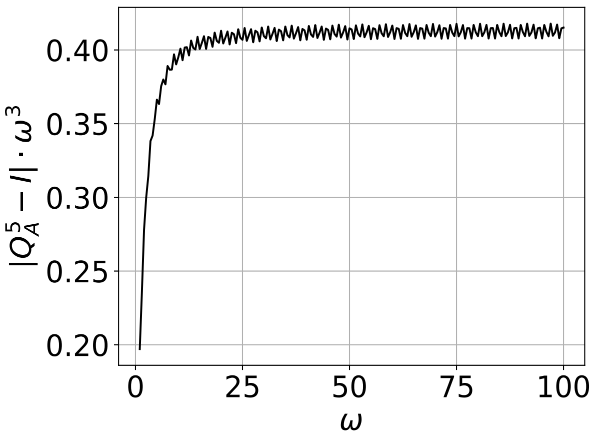
$$\int \sin(x) e^{i\omega(x-2)^2}\, dx = \frac{i\sqrt{\pi}}{4\sqrt{-i\omega}} \left[e^{\frac{i}{4\omega}(8\omega-1)} \operatorname{erf}\left(\frac{i}{2\sqrt{-i\omega}} (2\omega x - 4\omega + 1) \right) + \right.$$
$$\left. + e^{-\frac{i}{4\omega}(8\omega-1)} \operatorname{erf}\left(\frac{i}{2\sqrt{-i\omega}} (2\omega x - 4\omega + 1) \right) \right]. \tag{2.18}$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$.

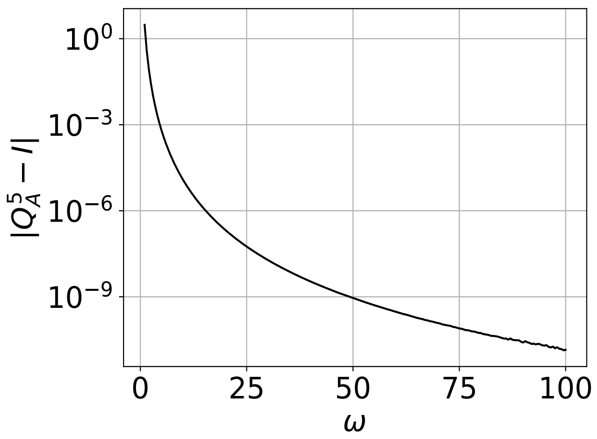
Now we compare the exact value of the integral with Q_2^A *and* Q_5^A.



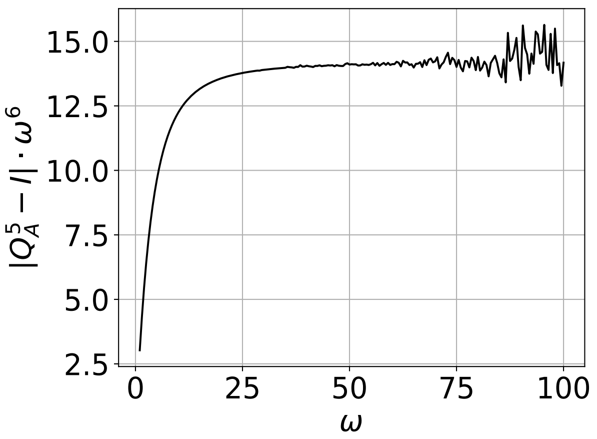
(2.2a) Absolute error on a logarithmic scale of using Q_2^A to evaluate $\int_{-1}^{1} \sin(x) e^{i\omega(x-2)^2}\, dx$.



(2.2b) Absolute error of using Q_2^A to evaluate $\int_{-1}^{1} \sin(x) e^{i\omega(x-2)^2}\, dx$ scaled by ω^3.



(2.3a) Absolute error on a logarithmic scale of using Q_5^A to evaluate $\int_{-1}^{1} \sin(x) e^{i\omega(x-2)^2}\, dx$.



(2.3b) Absolute error of using Q_5^A to evaluate $\int_{-1}^{1} \sin(x) e^{i\omega(x-2)^2}\, dx$ scaled by ω^6.

As we can see by comparing figures (2.2a) and (2.3a), the absolute error goes down 4 orders of magnitude when we go from Q_2^A to Q_5^A. Furthermore, both methods provide satisfactory results when looked at individually and follow the expected rate of error decay.

We also note that the integral in this example has a stationary point but it lies outside the domain of integration.

Lastly, we present an anti-example. We used the fact that $g'(x)$ has no stationary points in $[a, b]$ when deriving the methods but perhaps this method extends simply to the case where it does have stationary points in $[a, b]$. This turns out not to be the case.

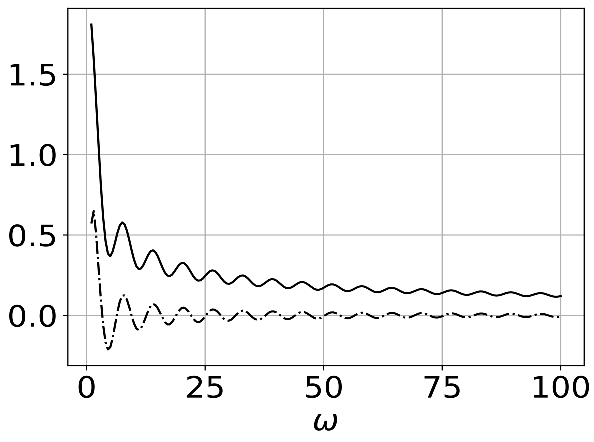
Example 2.1.4 *Consider the following very simple integral,*

$$I = \int_{-1}^{1} e^{i\omega x^2}\, dx \tag{2.19}$$

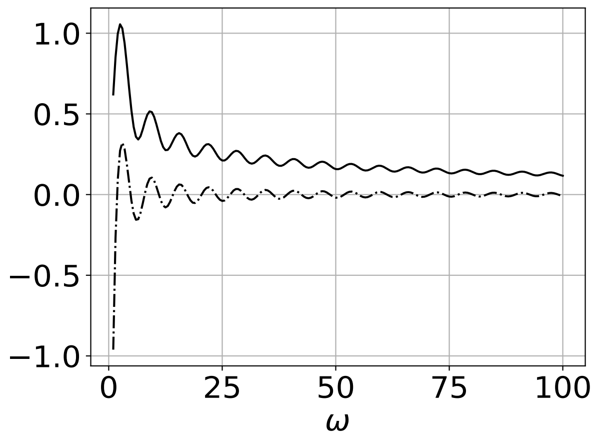
which has a closed form expression in terms of the special function erf(x)*,*

$$I = \frac{\sqrt{\pi}}{2\sqrt{-i\omega}}\left[\operatorname{erf}\left(\sqrt{-i\omega}\right) - \operatorname{erf}\left(-\sqrt{-i\omega}\right)\right]. \tag{2.20}$$

Let Q_2^A denote the asymptotic expansion of I truncated to 3 terms. If we compare the value of Q_2^A to the exact value over $0 \leq \omega \leq 100$*, it is clear that the method is not computing the correct value. This is shown in the plots (2.4a) and (2.4b).*



(2.4a) Plots of Re(I) (solid line) and Re(Q_2^A) (dotted line).



(2.4b) Plots of Im(I) (solid line) and Im(Q_2^A) (dotted line).

Furthermore, when we plot the error in figure (2.5), we notice that it is asymptotically $O(\omega^{-1/2})$ instead of $O(\omega^{-p-1})$.

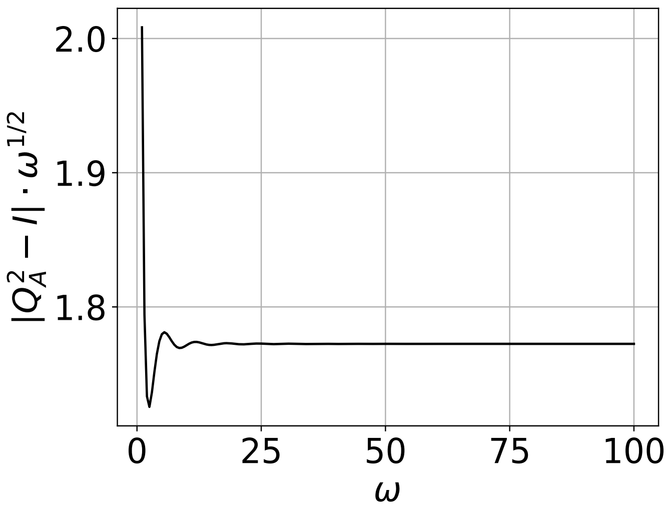


Figure 2.5: Absolute Error scaled by $\omega^{1/2}$.

Neither of these two failures can be simply remedied by calculating more terms of the expansion. Some more non-trivial work is needed to extend this method, which is described in the next section.

2.2 In The Presence of Stationary Points

Now we look at the general case. Without loss of generality[1], assume that $[a, b] = [0, 1]$ and that $g(x)$ has a single isolated rth order stationary point at $\xi \in [0, 1]$. That is,

$$g(\xi) = g'(\xi) = \ldots = g^{(r-1)}(\xi) = g^{(r)}(\xi) = 0 \text{ and } g^{(r+1)}(\xi) \neq 0. \tag{2.21}$$

The requirement that $g(\xi) = 0$ can be relaxed as if $g(\xi) \neq 0$, we can simply evaluate the following equivalent integral,

$$e^{i\omega g(\xi)} \int_a^b f(x) e^{i\omega g(x) - g(\xi)}\, dx.$$

We start by adding and subtracting the power series expansion of $f(x)$ around $x = \xi$ up to order $r - 1$ gives

$$\begin{aligned}
\int_0^1 f(x)e^{i\omega g(x)}\, dx &= \int_0^1 \left(f(x) + \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j - \sum_{j=0}^{r-1} \frac{f(\xi)}{j!}(x-\xi)^j \right) e^{i\omega g(x)}\, dx \\
&= \int_0^1 \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j e^{i\omega g(x)}\, dx + \int_0^1 \left(f(x) - \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j \right) e^{i\omega g(x)}\, dx \\
&= \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\, dx + \int_0^1 \left(f(x) - \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j \right) e^{i\omega g(x)}\, dx
\end{aligned}$$

Now we multiply the second term by $\dfrac{g'(x)i\omega}{g'(x)i\omega}$,

$$\begin{aligned}
&= \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\, dx + \\
&\quad + \frac{1}{i\omega} \int_0^1 \frac{\left(f(x) - \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j \right)}{g'(x)} i\omega g'(x) e^{i\omega g(x)}\, dx
\end{aligned} \tag{2.22}$$

And now we rewrite the last term in the second integral,

$$\begin{aligned}
\int_0^1 f(x)e^{i\omega g(x)}\, dx &= \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\, dx + \\
&\quad + \frac{1}{i\omega} \int_0^1 \frac{\left(f(x) - \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j \right)}{g'(x)} \frac{d}{dx}\left(e^{i\omega g(x)} \right) dx.
\end{aligned} \tag{2.23}$$

Keep in mind the identity (2.23). We will be using it over and over. As before, we are going to define the asymptotic expansion recursively. To that end, let

$$\rho_0[f](x) := f(x) \tag{2.24}$$

[1]We can do this because any interval $[a, b]$ can be linearly mapped to $[0, 1]$.

Now, starting with (2.23), we apply integration by parts to the second term.

$$\int_0^1 f(x)e^{i\omega g(x)}\,dx = \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx + \frac{e^{i\omega g(x)}}{i\omega} \frac{\left(\rho_0[f](x) - \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\Bigg|_0^1 + \\ - \frac{1}{i\omega}\int_0^1 \frac{d}{dx}\left[\frac{\left(\rho_0[f](x) - \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\right] e^{i\omega g(x)}\,dx \tag{2.25}$$

Expecting a pattern, let's define

$$\rho_1[f](x) := \frac{d}{dx}\left[\frac{\left(\rho_0[f](x) - \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\right] = \frac{d}{dx}\left[\frac{\left(f(x) - \sum_{j=0}^{r-1} \frac{f^{(j)}(\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\right]. \tag{2.26}$$

Then we can rewrite equation (2.25) as

$$\int_0^1 f(x)e^{i\omega g(x)}\,dx = \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx + \frac{e^{i\omega g(x)}}{i\omega} \frac{\left(\rho_0[f](x) - \sum_{j=0}^{r-1} \frac{\rho_0^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\Bigg|_0^1 + \\ - \frac{1}{i\omega}\int_0^1 \rho_1[f](x)e^{i\omega g(x)}\,dx. \tag{2.27}$$

The last term in (2.27) is exactly the same form as the left hand side of identity (2.23). So, using the identity to expand gives

$$-\frac{1}{i\omega}\int_0^1 \rho_1[f](x)e^{i\omega g(x)}\,dx = -\frac{1}{i\omega}\sum_{j=0}^{r-1} \frac{\rho_1^{(j)}[f](\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx + \\ - \frac{1}{(i\omega)^2}\int_0^1 \frac{\left(\rho_1[f](x) - \sum_{j=0}^{r-1} \frac{\rho_1^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)} \frac{d}{dx}\left(e^{i\omega g(x)}\right)\,dx. \tag{2.28}$$

From here, the pattern is clear. We alternate between using integration by parts to the last term of the expression and then follow that up with an application of the identity (2.23). Each time we apply the identity, we pick up a term of the form

$$\pm \frac{1}{(i\omega)^\alpha}\sum_{j=0}^{r-1} \frac{\rho_\alpha^{(j)}[f](\xi)}{j!} \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx. \tag{2.29}$$

Each application of integration by parts yields a term of the form

$$\pm \frac{e^{i\omega g(x)}}{(i\omega)^\alpha} \frac{\left(\rho_\alpha[f](x) - \sum_{j=0}^{r-1} \frac{\rho_\alpha^{(j)}[f](\xi)}{j!}(x-\xi)^j\right)}{g'(x)}\Bigg|_0^1. \tag{2.30}$$

With some careful book-keeping of the signs of each term and the powers of the coefficients $\frac{1}{i\omega}$, we arrive at the result. Let

$$\mu_j(\omega,\xi) = \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx, \quad j \geq 0; \tag{2.31}$$

$$\rho_0[f](x) = f(x), \tag{2.32}$$

$$\rho_{k+1}[f](x) = \frac{d}{dx}\left(\frac{\rho_k[f](x) - \sum_{j=0}^{r-1}\frac{\rho_k^{(j)}[f](\xi)}{j!}(x-\xi)^j}{g'(x)}\right) \quad k \geq 0 \tag{2.33}$$

Then the asymptotic expansion is

$$\begin{aligned}\int_0^1 f(x)e^{i\omega g(x)}\,dx \sim \sum_{j=0}^{r-1}\frac{1}{j!}\mu_j(\omega,\xi)\sum_{m=0}^{\infty}\frac{1}{(-i\omega)^m}\rho_m^{(j)}[f](\xi) \\ -\sum_{m=0}^{\infty}\left(\frac{e^{i\omega g(x)}}{(i\omega)^{m+1}g'(x)}(\rho_m[f](x)-\rho_m[f](\xi))\right)\Bigg|_0^1.\end{aligned} \tag{2.34}$$

And so by truncating the series to the first p terms, we obtain the asymptotic method for highly oscillatory integrals with an rth order stationary point $\xi \in [0,1]$:

$$\begin{aligned}Q_p^A[f,g] \equiv \sum_{j=0}^{r-1}\frac{1}{j!}\mu_j(\omega,\xi)\sum_{m=0}^{p-1}\frac{1}{(-i\omega)^m}\rho_m^{(j)}[f](\xi) \\ -\sum_{m=0}^{p-1}\left(\frac{e^{i\omega g(x)}}{(i\omega)^{m+1}g'(x)}(\rho_m[f](x)-\rho_m[f](\xi))\right)\Bigg|_0^1.\end{aligned} \tag{2.35}$$

With the expansion at hand, we can now comment on a few things. Firstly, if we have any finite number of stationary points in the interval, we can simply partition the interval in such a way that each partition only has one stationary point. Secondly, this method can be applied to any interval $[a,b]$, since we can simply transform the interval to $[0,1]$.

We need to compute the moments (2.31) in order to be able to use this method. These integrals are themselves oscillatory and in some sense, all we have done is reduced the problem of computing highly oscillatory integrals to computing highly oscillatory integrals with a slightly "nicer" function. Fortunately, these only depend on the function g and so can be precomputed, if the method is to be used repeatedly. Furthermore, as we are out to automate these methods, we will use Maple's arbitrary precision arithmetic capabilities in order to calculate these integrals precisely.

Another important thing to note here is that the computation of $\rho_k^{(j)}(x)$ requires the application of L'Hopital's rule repeatedly (as we need to evaluate $\rho_{k-1}^l(\xi)$, which by direct substitution leads to an indeterminate form). In the form it's presented, computation with this method for an arbitrary f and g would be a fairly tedious task. However, with the aid of Maple, we are able to symbolically find the limit and thus are able to automate this method as well for an arbitrary f and g.

Now let us derive the asymptotic error for this method. For this, we need a result from [22], which we state here with the appropriate translation of notation and convention.

Proposition 2.2.1 *For an integral $\int_a^b f(x)e^{i\omega g(x)}\,dx$, where $g(x)$ has an rth order stationary point $\xi \in (a,b)$ and $f(x)$ is supported in a small neighborhood around ξ, then the following asymptotic expansion holds:*

$$I[\omega] = \int f(x)e^{i\omega g(x)}\,dx \sim \sum_{j=0}^{\infty} \frac{a_j}{\omega^{\frac{j+1}{r+1}}} \tag{2.36}$$

in the sense that for all $N, r \in \mathbb{N}$,

$$\left(\frac{d}{d\omega}\right)^r \left[I[\omega] - \sum_{j=0}^{\infty} \frac{a_j}{\omega^{\frac{j+1}{r+1}}}\right] = O(\omega^{-r-(N+1)/k}) \quad \text{as } \omega \to \infty. \tag{2.37}$$

Furthermore, the coefficients a_j depend on only finitely many derivatives of the functions f and g.

While the proof in [22] is not constructive and only presents the expression for the first coefficient, it is shown in [19] that each coefficient a_j only depends on j derivatives of f at the stationary point[2] ξ.

Then the moments $\mu_j(\omega, \xi)$ have an expansion

$$\mu_j(\omega, \xi) \sim \sum_{k=0}^{\infty} \frac{a_k}{\omega^{\frac{k+1}{r+1}}} \tag{2.38}$$

We know that $\frac{d^k}{dx^k}(x-\xi)^j\Big|_{x=\xi} = 0$ for $k \le j-1$. So it follows that $a_k = 0$ for $k \le j-1$. Therefore,

$$\mu_j(\omega, \xi) \sim \sum_{k=j}^{\infty} \frac{a_k}{\omega^{\frac{k+1}{r+1}}} \sim O(\omega^{-(j+1)/(r+1)}). \tag{2.39}$$

The asymptotic estimate of the moments $\mu_j(\omega, \xi)$ is the last piece of information we need to describe the error, which readily follows by the order of ω in the leading terms of the difference below.

$$\begin{aligned}
\text{Absolute error} &= \left| \int_a^b f(x)e^{i\omega g(x)}\,dx - Q_p^A[f,g] \right| \\
&= \left| \frac{\mu_0(\omega,\xi)}{(i\omega)^{1/(r+1)}} + \cdots - \left(\frac{e^{i\omega g(x)}}{(i\omega)^{p+1} g'(x)} (\rho_p[f](x) - \rho_p[f](\xi)) \right)\Bigg|_0^1 - \cdots \right| \\
&\sim O(\omega^{-p-\frac{1}{r+1}}) \qquad \text{as } \omega \to \infty.
\end{aligned}$$

[2]We can also say more about how many derivatives of g at ξ are needed to calculate a_j but we don't need this information

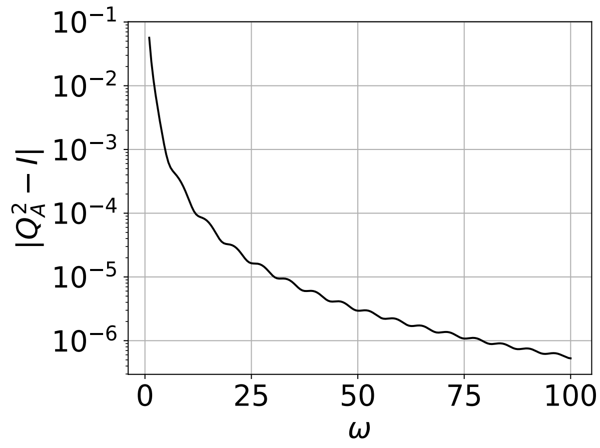
2.2.1 Numerical Examples

Example 2.2.1

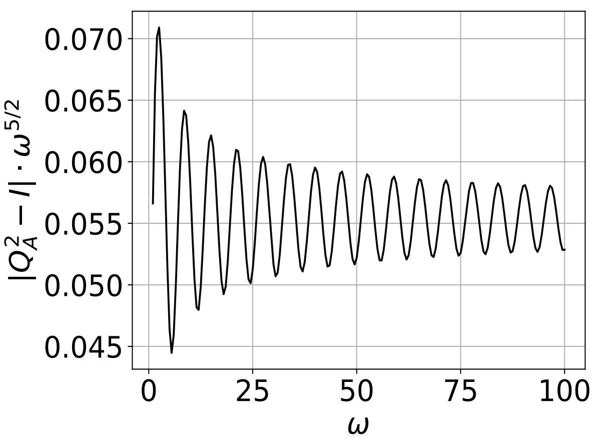
$$\int_{-1}^{1} \cos(x)e^{i\omega x^2}\, dx \tag{2.40}$$

The indefinite integral has a closed form expression in terms of erf,

$$\int \cos(x)e^{i\omega x^2}\, dx = \frac{\sqrt{\pi}e^{-i/4\omega}}{4\sqrt{-i\omega}}\operatorname{erf}\left(\sqrt{-i\omega}x - \frac{1}{2\sqrt{-i\omega}}\right) + \frac{\sqrt{\pi}e^{-i/4\omega}}{4\sqrt{-i\omega}}\operatorname{erf}\left(\sqrt{-i\omega}x + \frac{1}{2\sqrt{-i\omega}}\right). \tag{2.41}$$



(a) Log plot of the absolute error for the 2 term asymptotic method applied to integral (2.40).



(b) Absolute error scaled by $\omega^{5/2}$ for the 2 term asymptotic method applied to integral (2.40).

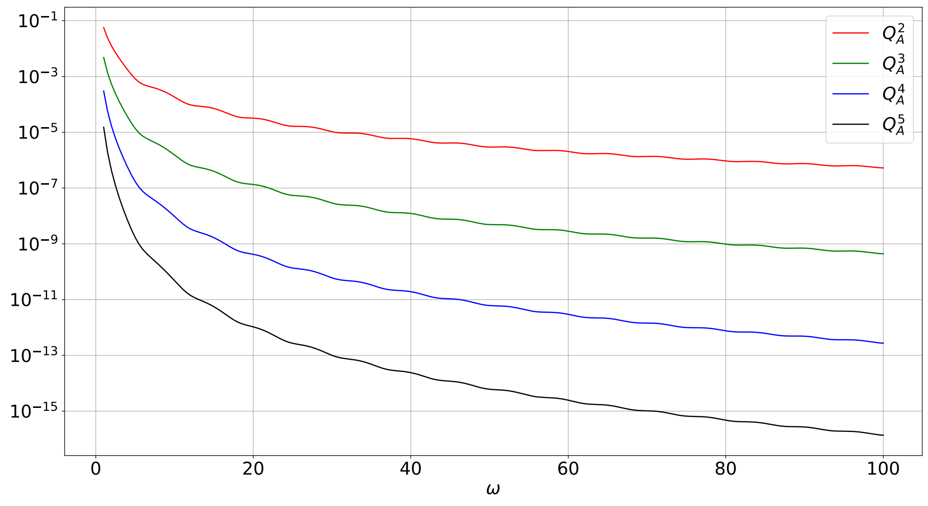


Figure 2.7: Log plot of the absolute error for the asymptotic method applied to integral (2.40) with increasing number of terms used.

Example 2.2.2 *As another example, consider the following integral,*

$$\int_{-1}^{1} \cos(x) e^{i\omega(7x^2+x^3)}\, dx \tag{2.42}$$

The function $g(x) = 7x^2 + x^3$ has a stationary point of order 2 at $x = 0$. We see in plot (2.8) that with just 3 terms, we attain an accuracy of 10 digits. Another interesting artifact that requires some explaining in the same plot is the strange oscillation in the blue curve. This is simply a result of the calculation reaching machine precision that introduces roundoff error in the intermediate steps, which result in the strange looking oscillation.

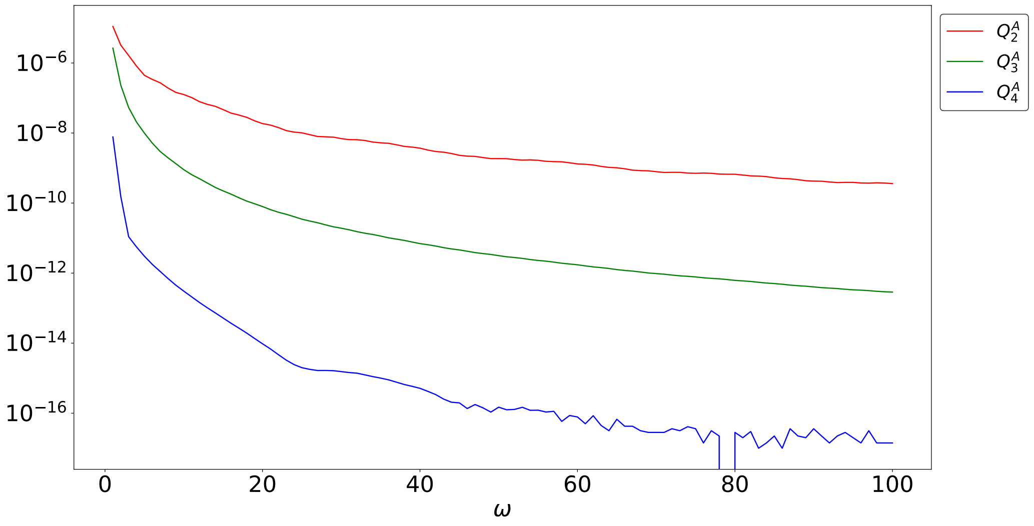


Figure 2.8: Absolute error for the 3,4 and 5 term asymptotic method applied to (2.42).

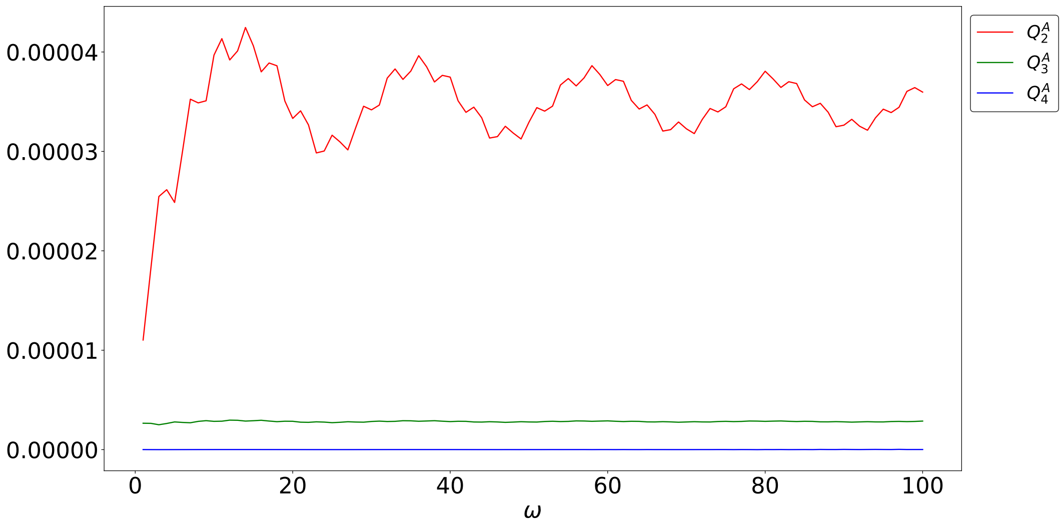


Figure 2.9: Absolute error scaled by $\omega^{p+1/2}$ for the 3,4 and 5 term asymptotic method applied to (2.42).

The asymptotic error in all three cases is $\omega^{-p-1/2}$ as we would expect it to be.

Chapter 3

Filon Type Methods

A large class of numerical quadrature methods are based on sampling the integrand at a set of points in the interval and interpolating this set of values in a convenient basis which is either easier to integrate or where the integral of the basis functions is known. A variation on this line of thinking will lead us to Filon type methods.

These methods were first presented in [5] and discussed extensively in the modern context in [13].

3.1 Method Derivation

Let $\{\tau_k\}_{k=0}^{N-1}$ be N distinct points in $[a, b]$ such that $a = \tau[0] < \tau[1] < \tau[2] < \ldots < \tau[N-1] = b$. One such choice of points is the Chebyshev-Lobatto nodes,

$$\{\tau_k\}_{k=0}^{n-1} = \left\{ \frac{a+b}{2} + \frac{(b-a)}{2} \cos\left(\frac{\pi(n-1)-k}{n-1} \right) \right\}_{k=0}^{n-1}.$$

Unless otherwise stated, we will always use the Chebyshev-Lobatto nodes for interpolation points.

Let $\{\phi_k(x)\}_{k=0}^{N-1}$ be a set of continuous functions from $[a, b]$ into $\mathbb{R}$ such that they satisfy the haar condition (they are linearly independent on $[a, b]$). This ensures that we can interpolate real continuous functions on $[a, b]$ in this basis and the interpolation is unique.

Now instead of interpolating the entire integrand $f(x)e^{i\omega g(x)}$ at the interpolation nodes, we interpolate only $f(x)$. Denote the interpolant of $f(x)$ by $\tilde{f}(x)$. The interpolant $\tilde{f}(x)$ is a function of the form

$$\tilde{f}(x) = \sum_{j=0}^{N-1} c_j \phi_j(x), \tag{3.1}$$

such that

$$\tilde{f}(\tau_\ell) = f(\tau_\ell), \qquad \text{for } \ell = 0, 1, \ldots, n-1. \tag{3.2}$$

Then the N-point Filon integration method is simply

$$Q_N^F[f,g] = \int_a^b \tilde{f}(x)e^{i\omega g(x)}\,dx. \tag{3.3}$$

At first glance, equation (3.3) does not appear to be progress. However, substituting the expression for $\tilde{f}(x)$ makes it clear that it is indeed a step forward.

$$\begin{aligned} Q_N^F[f,g] &= \int_a^b \tilde{f}(x)e^{i\omega g(x)}\,dx, \\ &= \int_a^b \sum_{j=0}^{N-1} c_j\phi_j(x)e^{i\omega g(x)}\,dx, \\ &= \sum_{j=0}^{N-1} c_j \int_a^b \phi_j(x)e^{i\omega g(x)}\,dx. \end{aligned} \tag{3.4}$$

And so if the integrals

$$\int_a^b \phi_j(x)e^{i\omega g(x)}\,dx \tag{3.5}$$

have a closed form expression or are precomputed numerically, then expression (3.4) is easy to evaluate. This captures the essence of Filon-type methods. The integrals (3.5) are called the moments.

The choice of the interpolation basis $\{\phi_k(x)\}$ depends on the function $g(x)$. Having said that, it is not true that we can always find a family of interpolating functions that makes the integrals (3.5) exactly computable. But for specific applications, we can always compute an expression for each of the integrals using the asymptotic-type methods we described in the previous section. Using asymptotic methods to evaluate the moments is an efficient route if $g(x)$ has no stationary points in $[a,b]$. However, the presence of stationary points would render this route more computationally expensive than evaluating the moments directly. This is owing to the fact that in the expansion (2.35), we need to calculate the integrals $\mu_j(\omega,\xi)$, which are themselves highly oscillatory.

In its most general form, we cannot make any general statements on the quality of the approximation derived from the Filon method. This is due to the fact that it is dependent on the quality on interpolation $\tilde{f}(x)$. For the rest of the chapter, we will look at the two most natural choices for interpolational basis functions, their error bounds and examples of the method at work.

3.2 Filon-Lagrange Method

Let's begin with one such choice for the interpolational basis (and perhaps the most used one), the monomial basis and Lagrange interpolation. The basis functions are monomials,

$$\{\phi_k\}_{k=0}^{N-1} = \{1, x, x^2, \ldots, x^{N-1}\}. \tag{3.6}$$

The interpolant $\tilde{f}(x)$ is then

$$\tilde{f}(x) = \sum_{k=0}^{N-1} c_k x^k,$$

where the c_k's are obtained by solving the system

$$\begin{bmatrix} f(a) \\ f(\tau_1) \\ \vdots \\ f(b) \end{bmatrix} = \begin{bmatrix} 1 & a & a^2 & \cdots & a^{N-1} \\ 1 & \tau_1 & \tau_1^2 & \cdots & \tau_1^{N-1} \\ \vdots & & \ddots & & \vdots \\ 1 & b & b^2 & \cdots & b^{N-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix}.$$

Let's examine the case when $g(x)$ has no stationary points in the interval $[a, b]$. Then this method has an error on the order of $O(\omega^{-2})$. This can be obtained as follows.

$$\text{Absolute Error} = \left| \int_a^b f(x) e^{i\omega g(x)}\, dx - \int_a^b \tilde{f}(x) e^{i\omega g(x)}\, dx \right|$$

$$\text{Absolute Error} = \left| \int_a^b (f(x) - \tilde{f}(x)) e^{i\omega g(x)}\, dx \right|$$

As g does not have stationary points in $[a, b]$, we have the asymptotic expansion (2.12)

$$\text{Absolute Error} = \left| -\sum_{m=0}^{\infty} \frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(b)}}{g'(b)} f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)} f_m(a) \right] \right|$$

$$\text{Absolute Error} = \left| \frac{1}{(i\omega)} \left[\frac{e^{i\omega g(b)}}{g'(b)} (f - \tilde{f})(b) - \frac{e^{i\omega g(a)}}{g'(a)} (f - \tilde{f})(a) \right] - \sum_{m=1}^{\infty} \frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(b)}}{g'(b)} f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)} f_m(a) \right] \right|$$

But we know that $f(a) = \tilde{f}(a)$ and $f(b) = \tilde{f}(b)$. So the first term in the above expression is 0. Thus we are left with

$$\text{Absolute Error} = \left| -\sum_{m=1}^{\infty} \frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(b)}}{g'(b)} f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)} f_m(a) \right] \right|$$

This shows that the error is $O(\omega^{-2})$.

At this point, it is worth noting that if we require the interpolant to match the derivatives of $f(x)$, we can improve this rate of error decay. This will be the subject of the next section. Now onto some demonstrations of this method.

Example 3.2.1 *Consider integrals of the form*

$$\int_a^b f(x) e^{i\omega x}\, dx, \tag{3.7}$$

where $f(x)$ is any continuous function.

We can use Filon-Lagrange to evaluate integrals of this form quite easily. This is due to the fact that the moments have closed form expressions, that is

$$\int x^n e^{i\omega x}\,dx = \frac{ix^n}{\omega}(n((n-1)! - \Gamma(n,-i\omega x))(-i\omega x)^{-n} - e^{i\omega x}),$$

where $\Gamma(a,x)$ *is the upper incomplete Gamma function, given by*

$$\Gamma(a,z) = \int_z^\infty t^{a-1}e^{-t}\,dt.$$

Both Maple and Python have very efficient internal numerical methods to compute values of $\Gamma(a,z)$*. With this integral identity at hand, we see that*

$$Q_N^F[f,x] = \int_a^b \tilde{f}(x)e^{i\omega x}\,dx$$
$$Q_N^F[f,x] = \int_a^b \sum_{j=0}^{N-1} c_j x^j e^{i\omega x}\,dx$$

where the c_j*'s are the Lagrange interpolation coefficients of* $f(x)$*. Then it is clear that*

$$Q_N^F[f,x] = \sum_{j=0}^{N-1} c_j \int_a^b x^j e^{i\omega x}\,dx$$
$$Q_N^F[f,x] = \sum_{j=0}^{N-1} c_j \left(\frac{ix^j}{\omega}(n((n-1)! - \Gamma(j,-i\omega x))(-i\omega x)^{-j} - e^{i\omega x})\right)\Bigg|_a^b. \tag{3.8}$$

So, using expression (3.8), we can evaluate any integral of the form (3.7). As an example, we evaluate the following integral

$$\int_{-1}^{1} \frac{1}{1+x^2}e^{i\omega x}\,dx. \tag{3.9}$$

Its exact value is given by the following expression

$$\frac{i}{2}\left[e^{-\omega}E_1(1,-\omega(ix+1)) - e^{\omega}E_1(1,-\omega(ix-1))\right].$$

Now using equation (3.8), we obtain the following results:

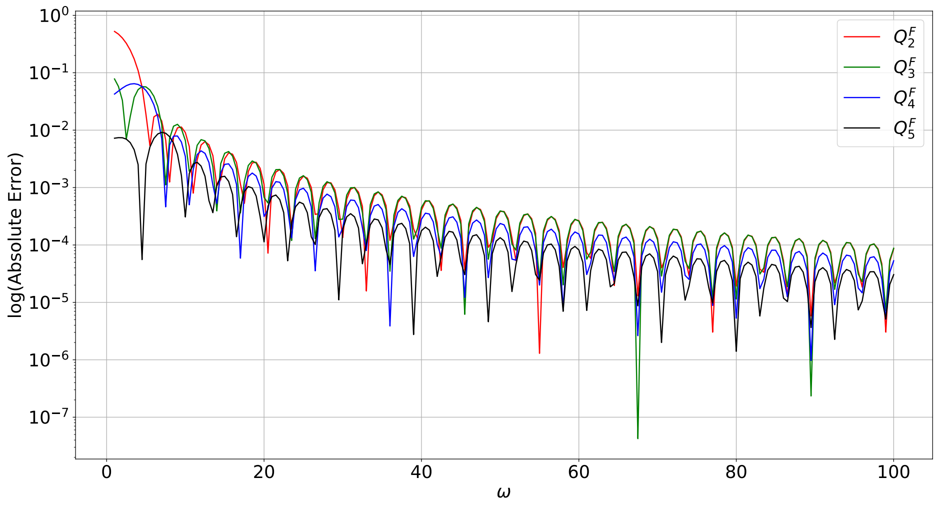


Figure 3.1: Absolute error for the Filon method plotted on a logarithmic scale for integral (3.9).

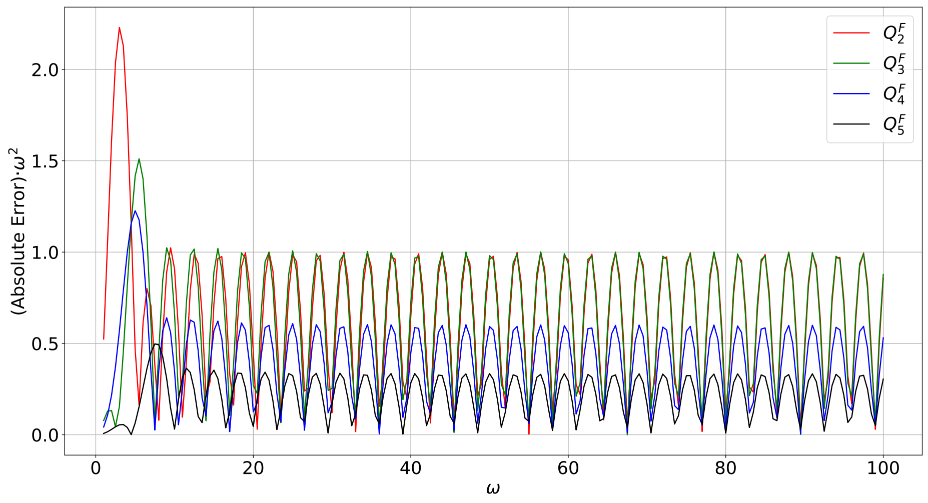


Figure 3.2: Absolute error scaled by ω^2 for the Filon method plotted on a logarithmic scale for integral (3.9).

In Figure (3.1) we see that the the error is indeed decreasing asymptotically and raising the number of interpolation nodes decreases the magnitude of the error. Furthermore, the error always decays like $\mathcal{O}(\omega^{-2})$ and adding interpolation points appears to not improve the result significantly. With further examples, we should expect to see the similar results.

Something to notice is that during the derivation process, at no point did we require the assumption that $g(x)$ has no stationary points in the integration domain. However, it will turn out that this method is inadequate in the presence of stationary points. We will show with an example that this asymptotic error bound is not valid in the stationary point case and the error will decay more slowly in the presence of stationary points. In fact, the higher the order of a stationary point, the slower the error will decay as $\omega \to \infty$ (this too will be demonstrated with an example).

Continuing on to another example, with a function $g(x)$ that is not simply x. Notice that $g(x)$ has a stationary point but it is outside the integration domain.

Example 3.2.2

$$\int_{-1}^{1} \sin(x)e^{i\omega(x-2)^2}\,dx \tag{3.10}$$

The closed form of this integral is

$$\begin{aligned}\int \sin(x)e^{i\omega(x-2)^2}\,dx =& \frac{i\sqrt{\pi}}{4\sqrt{-i\omega}}\left[e^{\frac{i}{4\omega}(8\omega-1)}\operatorname{erf}\left(\frac{i}{2\sqrt{-i\omega}}(2\omega x-4\omega+1)\right)+\right.\\ &\left.+e^{-\frac{i}{4\omega}(8\omega-1)}\operatorname{erf}\left(\frac{i}{2\sqrt{-i\omega}}(2\omega x-4\omega+1)\right)\right],\end{aligned} \tag{3.11}$$

where erf *is the error function as defined before.*

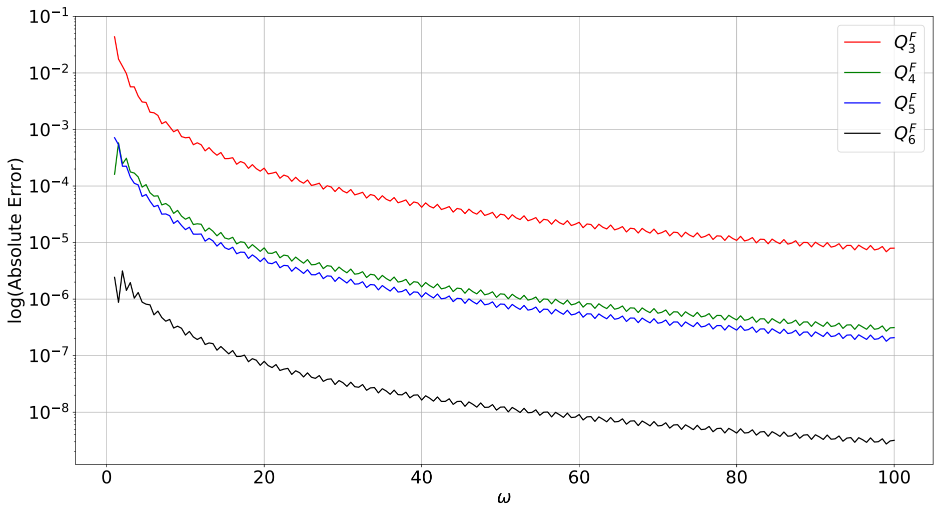


Figure 3.3: Log plot of the absolute error for 3-point Filon method used on integral (3.10).

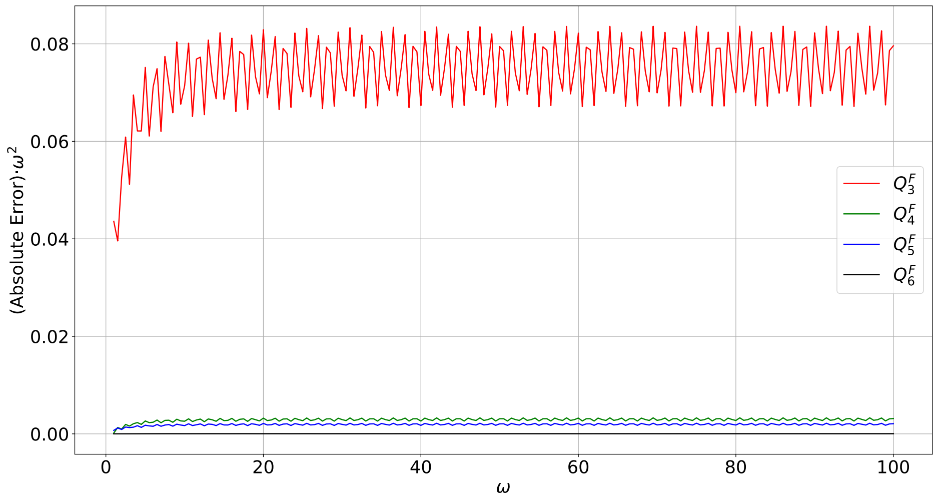


Figure 3.4: Scaled absolute error for 3-point Filon method used on integral (3.10).

From plots (3.3) and (3.4), we clearly see that the presence of a stationary point outside the integration domain has no effect on the method. The next example will show that the presence of a stationary point inside the integration domain does.

Example 3.2.3 *Consider the following family of integrals,*

$$I_n = \int_{-1}^{1} \cos(x) e^{i\omega x^{2n}}\, dx \quad \textit{for } n \geq 1 \tag{3.12}$$

We first use a Filon-Type method with 3 points and 4 points on I_1 and compare the results.

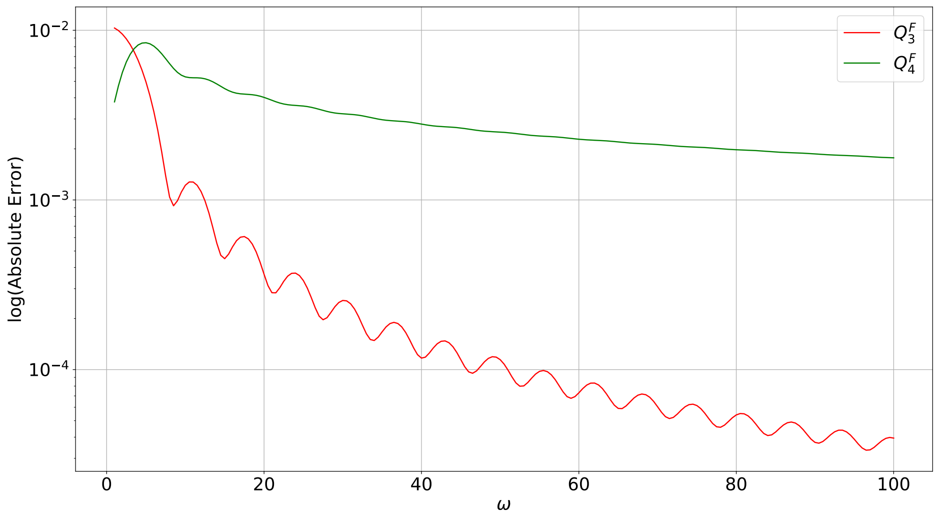
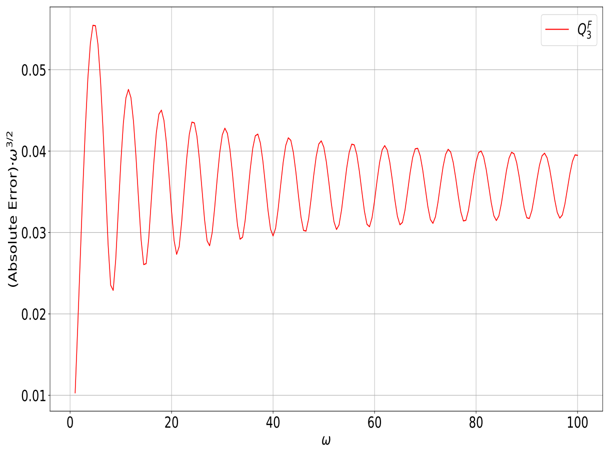


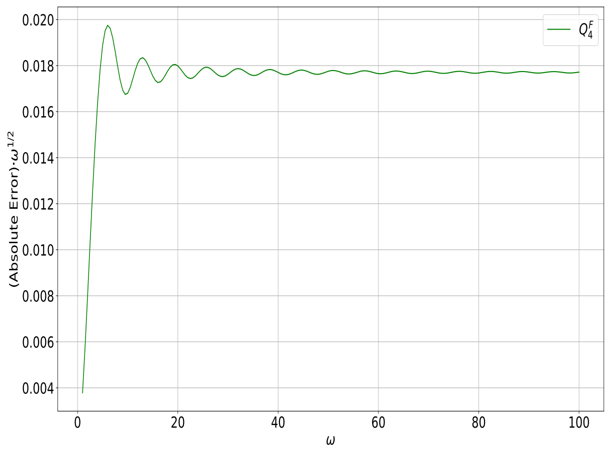
Figure 3.5: Comparison of the 3-point Filon method and the 4-point Filon method for integral I_1.

The resulting log error is plotted in Figure (3.5). The first thing that we notice is that the method with 3 points outperforms the method with 4 points. In fact, with a bit more expermi-nentation, it can be seen that the methods which have an odd number of interpolation points outperform their even numbered counterparts. This may appear strange at first but the rea-son for this is whenever we use an odd number of points, our stationary point is one of the interpolation points. We will use this fact later when we attempt to generalize this method.

Next we look at how the error behaves asymptotically.



(a) Scaled Error plot of the 3-point Filon method applied to I_1.



(b) Scaled Error plot of the 4-point Filon method applied to I_1.

In Figures (3.6a) and (3.6b) we see that neither the 3-point method nor the 4-point formula acheives the asymptotic error $O(\omega^{-2})$. The 3-point method achieves $O(\omega^{-3/2})$ and the 4-point method acheives $O(\omega^{-1/2})$. In I_1, the stationary point is of order 1. Let us now look at I_2, which has a stationary point of order 2.

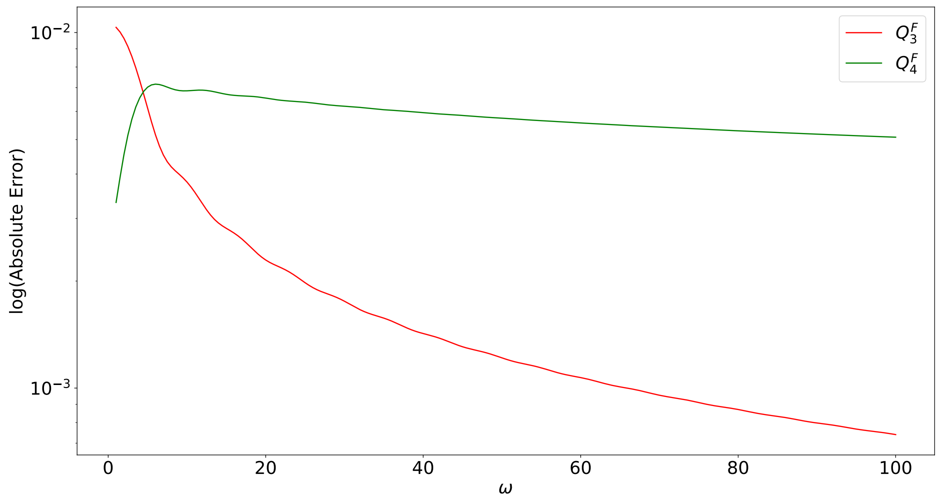
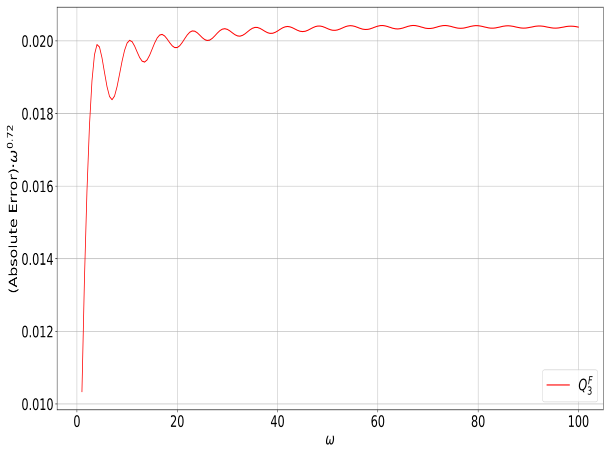
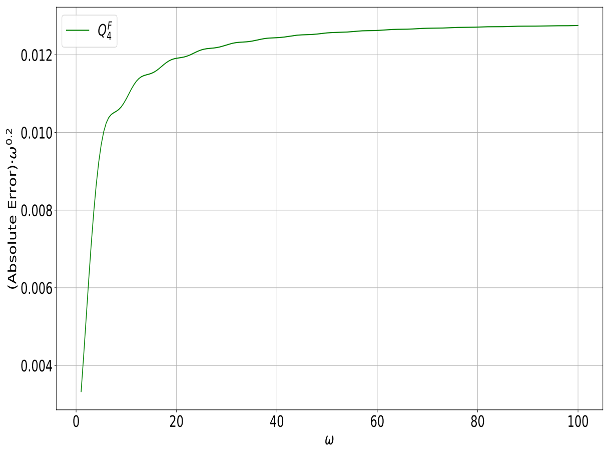


Figure 3.7: Comparison of the 3-point and 4-point Filon method applied to I_2.

It is clear from Figure (3.7) that the performance of the method is noticeably worse in the presence of a higher order stationary point. Not just do we get a greater magnitude, but also a slower asymptotic error decay, as can be seen in figures (3.8a) and (3.8b).



(a) Scaled error for the 3-point Filon method applied to I_2.



(b) Scaled error for the 4-point Filon method applied to I_2.

Example (3.2.3) forces us to conclude that this method is inadequate in the presence of stationary points. In the next section we will extend this method to cater to stationary points.

3.3 Filon-Hermite Method

Consider the integral,

$$\int_a^b f(x)e^{i\omega g(x)}\,dx, \tag{3.13}$$

where $g(x)$ has a stationary point $\xi \in (a,b)$ of order r. That is, $g'(\xi) = g''(\xi) = \ldots = g^{(r)}(\xi) = 0$ and $g^{(r+1)}(\xi) \neq 0$.

We attack this problem of extending the method by using the two facts we noted in the previous section:

1. If we require the interpolating function to match the derivatives of the function, we can lower the asymptotic error (both in magnitude and the rate at which is decays).

2. When the stationary point ξ is an interpolation point, we get a lower asymptotic error.

So, we require both of the above to hold.

Let $\{\tau_k\}_{k=0}^{N-1}$ be $N \geq 3$ distinct points in $[a,b]$ such that $a = \tau_0 < \tau_1 < \ldots < \tau_\nu = \xi < \ldots < \tau_{N-1} = b$. With each interpolation point τ_i, associate a confluency $s_i \in \mathbb{N}\backslash\{0,1\}$. So far we have the following two vectors,

$$\tau = [\tau_0, \tau_1, \ldots, \tau_{N-2}, \tau_{N-1}],$$
$$s = [s_0, s_1, \ldots, s_{N-2}, s_{N-1}].$$

At each node τ_i with confluency s_i, we match our interpolating polynomial up to the $(s_i - 1)$th derivative of f at that point. That is, we let $\tilde{f}(x)$ be a polynomial of order $(-1 + \sum_{i=0}^{N-1} s_i)$ such that it satisfies

$$\begin{aligned}
\tilde{f}(\tau_i) &= f(\tau_i)\\
\tilde{f}'(\tau_i) &= f'(\tau_i)\\
&\vdots\\
\tilde{f}^{(s_i-2)}(\tau_i) &= f^{(s_i-2)}(\tau_i)\\
\tilde{f}^{(s_i-1)}(\tau_i) &= f^{(s_i-1)}(\tau_i)
\end{aligned}$$

for $i = 0, 1, \ldots, N-1$. Then the Filon-Hermite method is simply

$$Q_{N,s}^{FH} = \int_a^b \tilde{f}(x)e^{i\omega g(x)}\,dx \tag{3.14}$$

where $s = \min(s_0, s_{N-1})$ and r is the order of the stationary point ξ.

We saw in the section (3.2) that the order of the stationary point has a direct impact on the asymptotic error achieved. It stands to reason that the number of derivatives we need to sample at the stationary point and at the other interpolation points will play a key role. However, before we attempt to quantify that, let us first check if this generalization does improve the performance in the presence of stationary points. We will do this by looking again at example

(3.2.3). Instead of matching only the function value, we also match the derivative value at each point as well.

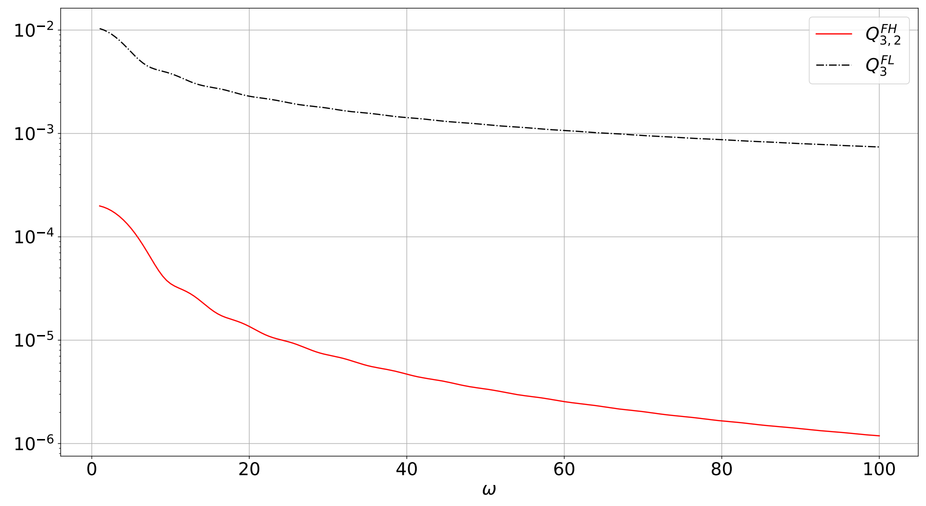


Figure 3.9: Absolute error plotted on a log scale for the Filon-Lagrange method with 3 nodes and the Filon-Hermite method with 3 nodes, each of confluency 2.

We see from the plot above that matching the first derivative does improve our results but keeps the asymptotic error the same. In fact, the promise of improved asymptotic error only holds simply when there is no stationary point[1]. In order to improve the asymptotic error in the presence of a stationary point, we need to sample further derivatives at the stationary point. The natural question to ask then is, how many derivatives does one need to sample in order to obtain a certain asymptotic error. This is what we look at now.

First, let us consider the case $g(x)$ does not have stationary points in $[a,b]$. Then we have at our disposal the asymptotic expansion (2.12),

$$I[f,\Omega] \sim \sum_{m=0}^{\infty} -\frac{1}{(i\omega)^{m+1}}\left[\frac{e^{i\omega g(x)}}{g'(x)} f_m(x)\right]\Bigg|_a^b = -\sum_{m=0}^{\infty}\frac{1}{(i\omega)^{m+1}}\left[\frac{e^{i\omega g(b)}}{g'(b)} f_m(b) - \frac{e^{i\omega g(a)}}{g'(a)} f_m(a)\right]$$

where

$$\begin{aligned} f_0(x) &= f(x), \\ \text{and } f_{m+1}(x) &= \frac{d}{dx}\left(\frac{f_m(x)}{g'(x)}\right). \end{aligned}$$

Using this simple Maple script, we can calculate expressions for f_m and collect the terms.

```
rho[0]:=f(x):
for k from 1 to 4 do
  rho[k]:=diff(rho[k-1]/diff(g(x),x),x);
```

[1]We will justify that this holds when we look at more numerical examples.

```
  print(collect(simplify(rho[k]),{seq(diff(f(x),[x$j]),j=0..k)
    }));
od:
```

If we look closely at the the first few $f_m(x)'s$, a pattern becomes apparent.

$$\begin{aligned}
f_0(x) &= f(x) \\
f_1(x) &= \frac{f'(x)g'(x) - g''(x)f(x)}{g'(x)^2} = \frac{1}{g'(x)}f'(x) + \frac{g''(x)}{g'(x)^2}f(x) \\
f_2(x) &= \frac{d}{dx}\left(\frac{f'(x)}{g'(x)^2}\right) + \frac{d}{dx}\left(\frac{g''(x)f(x)}{g'(x)^3}\right) \\
&= \frac{1}{g'(x)^2}f''(x) - 3\frac{g''(x)}{g'(x)^3}f'(x) + \frac{-g^{(3)}(x)g'(x) + 3g''(x)^2}{g'(x)^4}f(x) \\
&\vdots \\
f_m(x) &= \sum_{j=0}^{m} \sigma_j^m(x)\, f^{(j)}(x)
\end{aligned}$$

where each $\sigma_j^m(x)$ is of the form

$$\sigma_j^m(x) = \frac{\alpha_j(x)}{g'(x)^{2m-j}}$$

and $\alpha_j(x)$ depends on $g'(x), g''(x), \ldots, g^{(m+1-j)}(x)$. In particular, we always have

$$\sigma_m^m(x) = \frac{1}{g'(x)^m}$$

which is always non-zero by assumption.

With this re-expression of f_m's, we can rewrite the asymptotic expansion (2.12) as

$$I[f,\Omega] \sim \sum_{m=0}^{\infty} -\frac{1}{(i\omega)^{m+1}}\left[\frac{e^{i\omega g(x)}}{g'(x)}\sum_{j=0}^{m}\sigma_j^m(x)\, f^{(j)}(x)\right]\Bigg|_a^b. \tag{3.15}$$

With this fact in mind, let us attempt to calculate the absolute error of this method.

$$\begin{aligned}
\left|I[f,\omega] - Q_{N,s}^F H\right| &= \left|\int_a^b f(x)e^{i\omega g(x)}\,dx - \int_a^b \tilde{f}(x)e^{i\omega g(x)}\,dx\right| &(3.16)\\
&= \left|\int_a^b (f-\tilde{f})(x)e^{i\omega g(x)}\,dx\right| &(3.17)
\end{aligned}$$

And now using the expansion (3.15)

$$= \left| \sum_{m=0}^{\infty} -\frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(x)}}{g'(x)} \sum_{j=0}^{m} \sigma_j^m(x) \, (f - \tilde{f})^{(j)}(x) \right] \Bigg|_a^b \right| \tag{3.18}$$

$$= \left| \sum_{m=0}^{s-1} -\frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(x)}}{g'(x)} \sum_{j=0}^{m} \sigma_j^m(x) \, (f - \tilde{f})^{(j)}(x) \right] \Bigg|_a^b + \right.$$

$$\left. + \sum_{m=s}^{\infty} -\frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(x)}}{g'(x)} \sum_{j=0}^{m} \sigma_j^m(x) \, (f - \tilde{f})^{(j)}(x) \right] \Bigg|_a^b \right| \tag{3.19}$$

And as we have matched the function and its derivatives of $f(x)$ at the end point at least up to the $(s-1)$th order. So, $(f-\tilde{f})^{(j)}(a) = (f-\tilde{f})^{(j)}(b) = 0$ for $j = 0, 1, \ldots, s-1$. So, the first of the two sums is always 0. This leaves us with

$$= \left| \sum_{m=s}^{\infty} -\frac{1}{(i\omega)^{m+1}} \left[\frac{e^{i\omega g(x)}}{g'(x)} \sum_{j=0}^{m} \sigma_j^m(x) \, (f - \tilde{f})^{(j)}(x) \right] \Bigg|_a^b \right| \tag{3.20}$$

And from expression (3.20) we see that the asymptotic error is $O(\omega^{-s-1})$.

Let us take a pause here before continuing on to the general case and verify with an example the error is indeed $O(\omega^{-s-1})$.

Example 3.3.1 *Consider the integral*

$$\int_{-1}^{1} cos(x) e^{i\omega(x-2)^2} \, dx \tag{3.21}$$

If we use the Filon-Hermite method with 3 interpolation nodes and require that the interpolating function match the function and its derivative at each node, we expect the error to be $O(\omega^{-3})$. This is confirmed by plot (3.10).

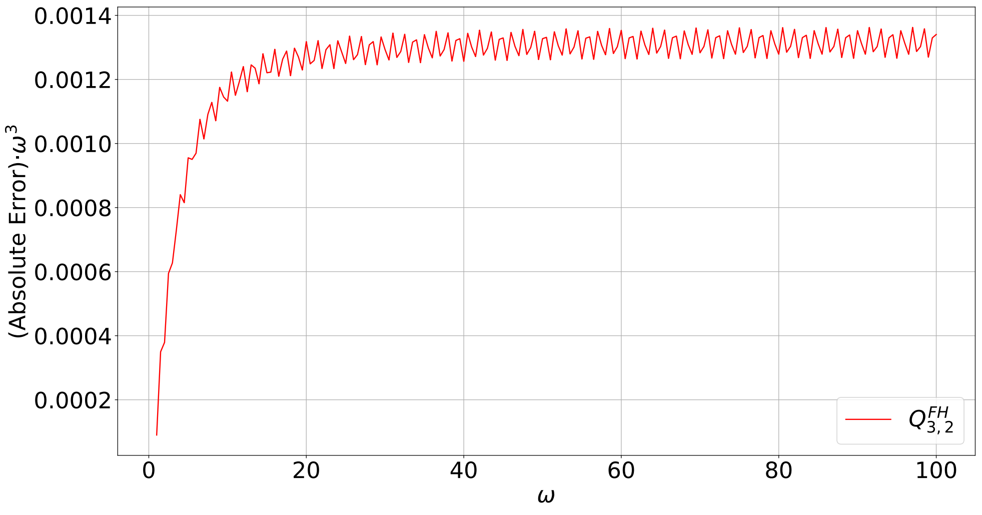


Figure 3.10: Absolute error scaled by ω^3 for the Filon-Hermite method with 4 nodes applied to (3.21), each of confluency 2.

A log plot of the absolute error confirms that we have 9 digits of accuracy by the time $\omega = 100$.

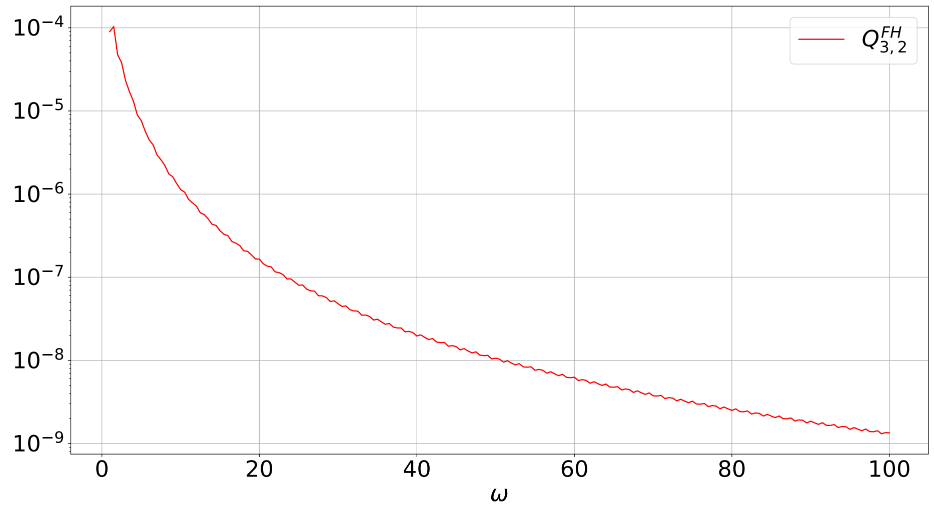


Figure 3.11: Absolute error on a log scale for the Filon-Hermite method with 4 nodes, each of confluency 2.

Now with that out of the way, let us consider the case where $g(x)$ has a stationary point in the integration domain.

The approach we will take for this is along the same lines as what we did for the previous case. Briefly put, we will first find out how many derivatives we need to sample at the stationary point in order to compute ρ_k's in the asymptotic expansion (2.31). Using this, we will be able to figure out exactly how many derivatives we need to sample at $x = \xi$ in order to match the accuracy of the Asymptotic method in the presence of stationary points, $O(\omega^{-s-1/(r+1)})$.

Assume that we have the integral,

$$\int_0^1 f(x)e^{i\omega g(x)}\,dx \tag{3.22}$$

with $g(x)$ having a stationary point of order r at $x = \xi \in (0,1)$. Recall that in the presence of stationary points, we have at our disposal the following asymptotic expansion,

$$\int_0^1 f(x)e^{i\omega g(x)}\,dx \sim \sum_{j=0}^{r-1} \frac{1}{j!}\mu_j(\omega,\xi) \sum_{m=0}^{\infty} \frac{1}{(-i\omega)^m}\rho_m^{(j)}[f](\xi) \\ - \sum_{m=0}^{\infty}\left(\frac{e^{i\omega g(x)}}{(i\omega)^{m+1}g'(x)}(\rho_m[f](x) - \rho_m[f](\xi))\right)\Bigg|_0^1.$$

where

$$\mu_j(\omega,\xi) = \int_0^1 (x-\xi)^j e^{i\omega g(x)}\,dx, \quad j \geq 0;$$

$$\rho_0[f](x) = f(x), \tag{3.23}$$

$$\rho_k[f](x) = \frac{d}{dx}\left(\frac{\rho_{k-1}[f](x) - \sum_{j=0}^{r-1} \frac{\rho_{k-1}^{(j)}[f](\xi)}{j!}(x-\xi)^j}{g'(x)}\right), \qquad k \geq 0. \tag{3.24}$$

In order to be able to calculate an expression any $\rho_k[f](x)$, we have to be able to evaluate $\rho_{k-1}[f](\xi),\ \rho'_{k-1}[f](\xi),\ \ldots,\ \rho_{k-1}^{(r-1)}[f](\xi)$. And as we have a stationary point at ξ, we need to verify that this these expressions are indeed always well-defined.

Away from $x = \xi$, the function $\rho_k[f](x)$'s are well defined[2] and evaluating them at a fixed $x = \tau_i$ just gives a linear combination of $f(\tau_i), f'(\tau_i), \ldots, f^{(k)}(\tau_i)$.

Let's focus on $x = \xi$. We can rewrite $f(x)$ and $g(x)$ as

$$f(x) = \sum_{j=0}^{\infty} \frac{f_j}{j!}(x-\xi)^j \quad \text{and} \quad g(x) = \sum_{j=r+1}^{\infty} \frac{g_j}{j!}(x-\xi)^j. \tag{3.25}$$

Then we have that $\rho_0[f](\xi) = f_0$ and $\rho_0^{(j)}[f](\xi) = f_j$. With this in mind,

$$\rho_1[f](x) = \frac{d}{dx}\left(\frac{f(x) - \sum_{j=0}^{r-1} \frac{f_j}{j!}(x-\xi)^j}{g'(x)}\right)$$

$$\rho_1[f](x) = \frac{d}{dx}\left(\frac{\sum_{j=0}^{\infty} \frac{f_j}{j!}(x-\xi)^j - \sum_{j=0}^{r-1} \frac{f_j}{j!}(x-\xi)^j}{g'(x)}\right)$$

$$\rho_1[f](x) = \frac{d}{dx}\left(\frac{\sum_{j=r}^{\infty} \frac{f_j}{j!}(x-\xi)^j}{g'(x)}\right)$$

$$\rho_1[f](x) = \frac{\sum_{j=r}^{\infty} \frac{f_j}{(j-1)!}(x-\xi)^{j-1}}{g'(x)} - \frac{g''(x)\sum_{j=r}^{\infty} \frac{f_j}{j!}(x-\xi)^j}{g'(x)^2} \tag{3.26}$$

And so evaluating (3.26) at $x = \xi$ is obtained by taking the limit as $x \to \xi$. We take the limits for each of the two terms separately for simplicity.

$$\lim_{x\to\xi} \frac{\sum_{j=r}^{\infty} \frac{f_j}{(j-1)!}(x-\xi)^{j-1}}{g'(x)} = \lim_{x\to\xi} \frac{\sum_{j=r}^{\infty} \frac{f_j}{(j-1)!}(x-\xi)^{j-1}}{\sum_{j=r}^{\infty} \frac{g_{j+1}}{j!}(x-\xi)^j} \tag{3.27}$$

[2] as long as we can evaluate $\rho_{k-1}[f](\xi),\ \rho'_{k-1}[f](\xi),\ \ldots,\ \rho_{k-1}^{(r-1)}[f](\xi)$

This is a $0/0$ indeterminate form. Applying L'Hopital's rule r times gives

$$\lim_{x\to\xi}\frac{\sum_{j=r}^{\infty}\frac{f_j}{(j-1)!}(x-\xi)^{j-1}}{g'(x)}=\lim_{x\to\xi}\frac{\frac{d^r}{dx^r}\left(\sum_{j=r}^{\infty}\frac{f_j}{(j-1)!}(x-\xi)^{j-1}\right)}{\frac{d^r}{dx^r}\left(\sum_{j=r}^{\infty}\frac{g_{j+1}}{j!}(x-\xi)^{j}\right)}$$
$$=\lim_{x\to\xi}\frac{\sum_{j=0}^{\infty}\frac{f_{j+1+r}}{j!}(x-\xi)^{j}}{\sum_{j=0}^{\infty}\frac{g_{j+1+r}}{j!}(x-\xi)^{j}}=\lim_{x\to\xi}\frac{f_{r+1}}{g_{r+1}}+O(x-\xi)=\frac{f_{r+1}}{g_{r+1}}\tag{3.28}$$

And now for the second term we have

$$\lim_{x\to\xi}-\frac{g''(x)\sum_{j=r}^{\infty}\frac{f_j}{j!}(x-\xi)^j}{g'(x)^2}=-\lim_{x\to\xi}\frac{\left(\sum_{j=r-1}^{\infty}\frac{g_{j+2}}{j!}(x-\xi)^j\right)\left(\sum_{j=r}^{\infty}\frac{f_j}{j!}(x-\xi)^j\right)}{\left(\sum_{j=r}^{\infty}\frac{g_{j+1}}{j!}(x-\xi)^j\right)^2}$$

Which we evaluate with Maple to get

$$=-\lim_{x\to\xi}\frac{(g_{r+2}f_r+f_{1+r}g_{1+r})r+g_{r+2}f_r}{(1+r)g_{1+r}^2}+O(x-\xi)$$
$$=-\frac{(g_{r+2}f_r+f_{1+r}g_{1+r})r+g_{r+2}f_r}{(1+r)g_{1+r}^2}\tag{3.29}$$

And so combining both limits yields,

$$\rho_1[f](x)=-\frac{1}{r+1}\frac{g_{r+2}}{g_{r+1}}f_r+\frac{1}{r+1}\frac{f_{r+1}}{g_{r+1}}+O(x-\xi)\tag{3.30}$$

From equation (3.30) we note that $\rho_1[f](\xi)$ depends linearly on $f^{(r)}(\xi)$ and $f^{(r+1)}(\xi)$. Furthermore we can say that $\rho_1^{(m)}[f](\xi)$ depends linearly on $f^{(r)}(\xi), f^{(r+1)}(\xi), \ldots, f^{(r+m+1)}(\xi)$ because in order to calculate it, we would take the mth derivative of (3.26), which will introduce m further derivatives of f. Using this inductive process, we can find a Taylor expansion of $\rho_1[f](x)$ around $x=\xi$,

$$\rho_1[f](x)=\sum_{m=0}^{\infty}\frac{\tilde{f}_m}{m!}(x-\xi)^m\tag{3.31}$$

where each $\tilde{f}_m$ depends linearly on $f^{(r)}(\xi), f^{(r+1)}(\xi), \ldots, f^{(r+m+1)}(\xi)$.

We can now use the same inductive argument to make a general statement. That is, $\rho_2[f](\xi)$ will depend linearly on $\tilde{f}_r$ and $\tilde{f}_{r+1}$, which means it depends linearly on $f^{(r)}(\xi), f^{(r+1)}(\xi), \cdots,$ $f^{(r+(r+1)+1)}(\xi)=f^{(2r+2)}(\xi)$.

And in general, for $k\geq 1$, $\rho_k[f](\xi)$ will depend linearly on $f^{(r)}(\xi), f^{(r+1)}(\xi), \cdots, f^{(k(r+1))}(\xi)$. So in order to calculate $\rho_k[f](\xi)$, we need to sample $k\cdot(r+1)$ derivatives at $x=\xi$.

This is the last piece of information we need to make a complete statement. Let $s=\min(s_0, s_{N-1})$ and $s_v=s\cdot(r+1)$. Then the error is asymptotically

$$|I-Q_N^{FH}|\sim O(\omega^{-s-\frac{1}{r+1}})\quad\text{as }\omega\to\infty.\tag{3.32}$$

This readily follows from substituting $(f-\tilde{f})(x)$ into the asymptotic expansion (2.31) and seeing that the lowest order ω term is $\omega^{-s-\frac{1}{r+1}}$. Let us now see this method in action.

Example 3.3.2 *Consider integrals of the form,*

$$\int_{-1}^{1} f(x)e^{i\omega x^r}\,dx \tag{3.33}$$

for $r \geq 2$. The function $g(x)$ has a stationary point of order $r-1$ at $x=0$, so we will need to sample appropriately.

Using Mathematica, we are able to compute an expression for the moments,

$$\int x^k e^{i\omega x^r}\,dx = -\frac{x^{1+m}}{r}(-i\omega x^r)^{-\frac{1+m}{r}} \cdot \Gamma\left(\frac{1+m}{r}, -i\omega x^r\right) \tag{3.34}$$

where $\Gamma(a,z)$ is the incomplete gamma function.

And finally, as the order of the stationary point is $r-1$, we require the interpolating polynomial to match $f(x)$ at $x=0$ upto the $(s\cdot r)$th derivative, where $s = \min\{s_0, s_{N-1}\}$.

$$Q_{N,s}^{FH} = \int_{-1}^{1} \tilde{f}(x)e^{i\omega g(x)} = \sum_{k=0}^{order(\tilde{f})} c_k \cdot \left(-\frac{x^{1+m}}{r}(-i\omega x^r)^{-\frac{1+m}{r}} \cdot \Gamma\left(\frac{1+m}{r}, -i\omega x^r\right)\right)\Bigg|_{-1}^{1} \tag{3.35}$$

where the c_k's are the interpolation coefficients for the expression of the interpolating polynomial in the monomial basis.

Let's see how the method performs for some fixed $f(x)$. Let's take the integral from Example 2.2.1,

$$\int_{-1}^{1} \cos(x)e^{i\omega x^2}\,dx.$$

The resulting error plots are shown in Figures (3.12, 3.13). As we increase the number of interpolation nodes, the absolute error drops.

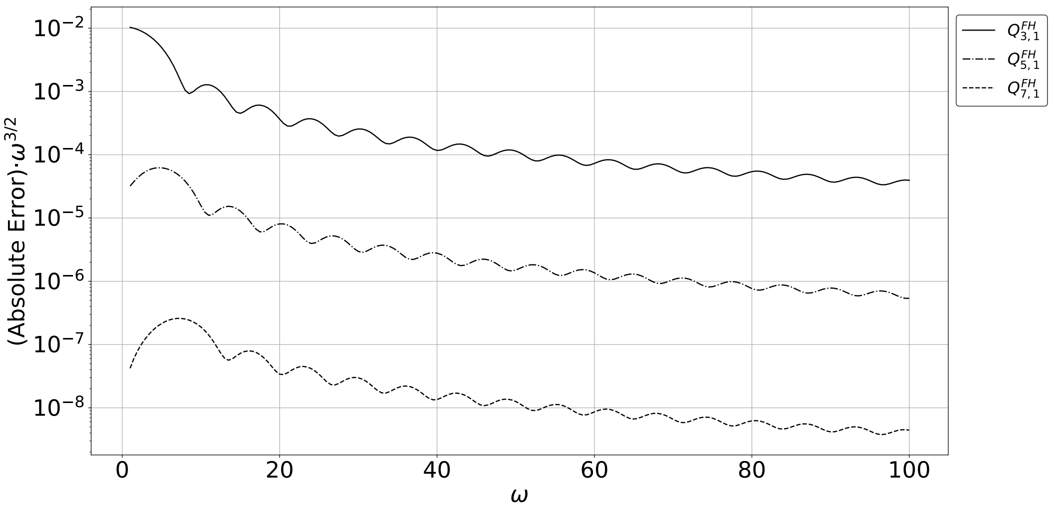


Figure 3.12: Error plot on the log scale with 3,5 and 7 interpolation nodes. The confluency all of the nodes is set to 1 and the stationary point node's is set to $s \cdot (r+1) = 2$.

Next we see in Figure (3.13) that the asymptotic error is what we expect it to be.

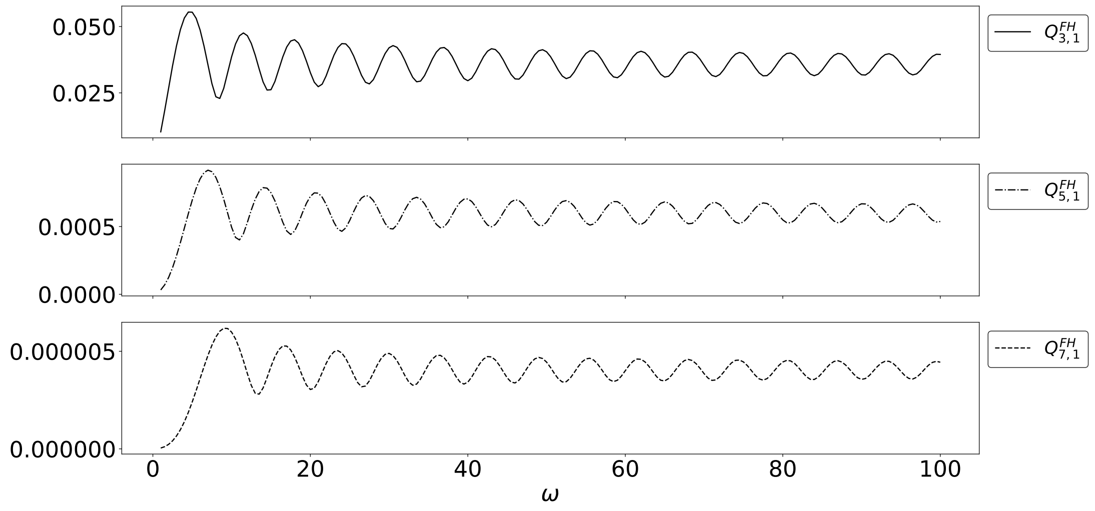


Figure 3.13: Absolute error scaled by $w^{3/2}$ for 3,5 and 7 interpolation nodes.

Now we raise the confluency of the nodes to see if the asymptotic error responds appropriately.

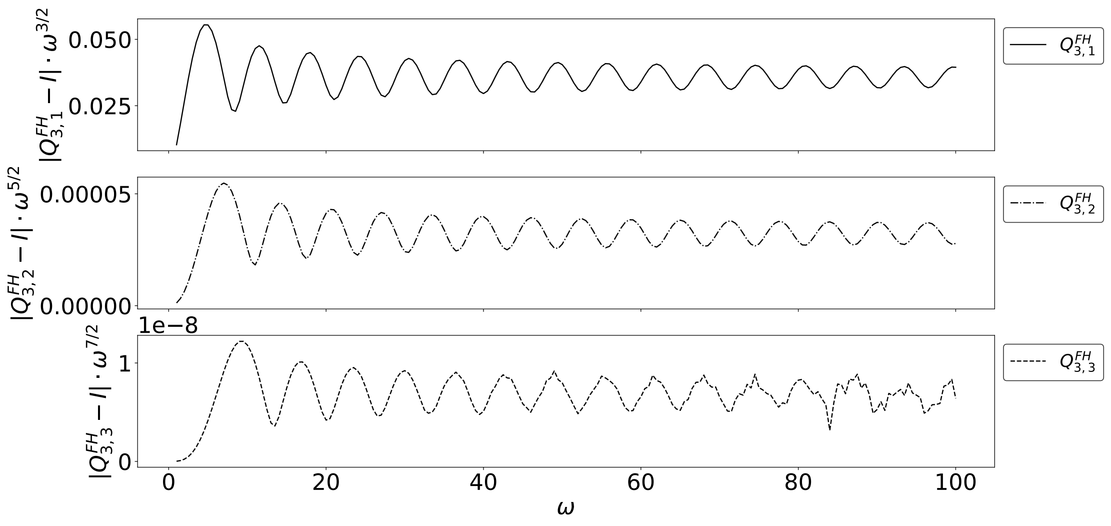


Figure 3.14: Absolute error scaled by $w^{s+1/2}$, for confluency values $s = 1, 2, 3$.

And as we see from the scaled error plots in Figure (3.14), the asymptotic error is $O(\omega^{s+\frac{1}{2}})$.

Before moving on to the next chapter, let's take a brief pause to collect the results from this chapter:

- In the absence of stationary points, the Filon-Hermite method will attain an asymptotic error of

$$O(\omega^{-\min(s_0,s_N)-1}).$$

That is, the asymptotic error depends only on how many derivatives we sample at the end points.

- In the presence of a stationary point, we can attain the same asymptotic error if we sample the function at the stationary point upto the $(s\cdot(r+1))$th derivative, where $s = \min(s_0, s_N)$ and r is the order of the stationary point.

- An observation to make is that the Filon method generally provides a better approximation than the asymptotic method but at the same time is a lot more computationally costly (at least in the absence of stationary points).

- Maple implementations of the methods described in this section can be found in Appendix (A.2.1) and (A.2.2).

Chapter 4

Moment Free Methods

The methods described in sections (2.2) and (3.3) for highly oscillatory integrals with stationary points all required us to evaluate moments numerically. While sometimes we can evaluate them explicitly, in general we are forced to rely on the internal quadrature method of whatever system we are implementing the methods. This is a recipe for disaster as we are trading in a highly oscillatory integral for a few slightly easier highly oscillatory integrals. Ideally, this is something one would want to avoid. The methods presented here allow us to do just that. The two methods discussed here were first presented in [20].

We will focus our attention on the integral

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx \tag{4.1}$$

where $g(x)$ has a stationary point of order r at $x = 0$. Furthermore, we require $g'(x), g''(x) \neq 0$ for $0 < |x| \leq 1$ and $g^{(r+1)}(0) > 0$. The reason why we are choosing to restrict our attention to $[-1, 1]$ will become clear shortly.

Both the methods rely on the creation of a basis, $\psi_{r,k}(x)$, which depends on the function $g(x)$ with the special property that the moments

$$\int_{-1}^{1} \psi_{r,k}(x)e^{i\omega g(x)}\,dx \tag{4.2}$$

can be calculated exactly. This means we always know the expression for the moments in closed form and so don't need to approximate them using classical quadrature.

As a first step, consider the special case where $\psi_{r,k}(x) = x^k$ and $g(x) = x^r$. If the corresponding moment (4.2) has a closed form solution, then there exists a continuous function $F(x)$ such that

$$\int x^k e^{i\omega x^r}\,dx = F(x)e^{i\omega x^r}$$

Differentiating both sides with respect to x yields

$$\begin{aligned} x^k e^{i\omega x^r} &= \frac{d}{dx}\left(F(x)e^{i\omega x^r}\right) \\ x^k &= F'(x) + i\omega r x^{r-1}F(x) \end{aligned} \tag{4.3}$$

The differential equation (4.3) has a known solution,

$$F(x) = \frac{\omega^{-\frac{1+k}{r}}}{r} e^{-i\omega x^r + \frac{1+k}{2r} i\pi} \left[\Gamma\left(\frac{1+k}{r}, -i\omega x^r\right) - \Gamma\left(\frac{1+k}{r}, 0\right) \right] \tag{4.4}$$

where $\Gamma(a, x)$ is the incomplete gamma function defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1}\, dt.$$

In the expression (4.4), we notice that x^r occurs several times. Define a new family of functions, $\phi_{r,k}$ obtained by replace x^r with $g(x)$ in (4.4) and an additional term to account for the branch cuts,

$$\phi_{r,k}(x) = D_{r+1,k}(\mathrm{sgn}(x)) \frac{\omega^{-\frac{1+k}{r+1}}}{r+1} e^{-i\omega g(x) + \frac{1+k}{2(r+1)} i\pi} \left[\Gamma\left(\frac{1+k}{r+1}, -i\omega g(x)\right) - \Gamma\left(\frac{1+k}{r+1}, 0\right) \right], \tag{4.5}$$

where

$$D_{r,k}(\mathrm{sgn}(x)) = \begin{cases} (-1)^k & x < 0 \text{ and } r \text{ even} \\ (-1)^k e^{-\frac{1+k}{r} i\pi} & x < 0 \text{ and } r \text{ odd} \\ -1 & \text{otherwise.} \end{cases} \tag{4.6}$$

Following the same train of thought, define the generalization of the differential equation (4.3),

$$\psi_k(x) = F'(x) + i\omega g'(x) F(x) \tag{4.7}$$

Then it stands to reason that substituting the equation (4.5) into the right hand side of the differential equation (4.7) should yield a basis in which the moment integrals always have closed form expressions. This yields the following g-dependent but ω-independent interpolation basis,

$$\mathcal{L}[\phi_{r,k}](x) = \mathrm{sgn}(x)^{r+k+1} \frac{|g(x)|^{\frac{k+1}{r}-1} g'(x)}{r}. \tag{4.8}$$

The details of the substitution and simplification can be found in the original paper [20] and are not included here as they are not relevant to the method itself. We have included some Maple code in appendix (A.4.1) if the reader wishes to verify that the expression above satisfies the differential equation for a given $g(x)$. This code also provides a good way to determine how well this method will perform for a given $g(x)$.

4.1 Moment-Free Asymptotic Method

Once we have obtained the basis $\{\phi_{r,k}(x)\}$, we obtain a new method by simply replacing the monomial basis terms in the classical asymptotic method derivation in section (2.2) with this new basis.

Define

$$\mu[f](x) = \sum_{k=0}^{r-2} c_k \phi_{r,k}(x) \tag{4.9}$$

so that

$$\mathcal{L}[\mu[f]](0) = f(0), \mathcal{L}[\mu[f]]'(0) = f'(0), \dots, \mathcal{L}[\mu[f]]^{(r-2)}(0) = f^{(r-2)}(0). \tag{4.10}$$

Then we can write integral (4.1) as

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx = \int_{-1}^{1} (f(x) + \mathcal{L}[\mu[f]](x) - \mathcal{L}[\mu[f]](x))e^{i\omega g(x)}\,dx \tag{4.11}$$

$$= \int_{-1}^{1} \mathcal{L}[\mu[f]](x)e^{i\omega g(x)}\,dx + \int_{-1}^{1} (f(x) - \mathcal{L}[\mu[f]](x))e^{i\omega g(x)}\,dx \tag{4.12}$$

The first integral above can be computed exactly,

$$= \sum_{k=0}^{r-2} c_k \cdot I\left[\mathcal{L}[\phi_{r,k}](x)\right] + \int_{-1}^{1} (f(x) - \mu[f](x))e^{i\omega g(x)}\,dx \tag{4.13}$$

Substituting in (**??**), the exact expression for $I\left[\mathcal{L}[\phi_{r,k}](x)\right]$,

$$= \sum_{k=0}^{r-2} c_k \cdot (\phi_{r,k}(1)e^{i\omega g(1)} - \phi_{r,k}(-1)e^{i\omega g(-1)}) + \int_{-1}^{1} (f(x) - \mu[f](x))e^{i\omega g(x)}\,dx$$

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx = \mu[f](1)e^{i\omega g(1)} - \mu[f](-1)e^{i\omega g(-1)} + \frac{1}{i\omega}\int_{-1}^{1} \frac{(f(x) - \mu[f](x))}{g'(x)}(i\omega g'(x)e^{i\omega g(x)})\,dx \tag{4.14}$$

The integral in the expression above has a removable singularity at $x = 0$. So, we can use integration by parts there. We simultaneously relabel $f(x)$ to $\sigma_0(x)$.

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx = \mu[\sigma_0](1)e^{i\omega g(1)} - \mu[\sigma_0](-1)e^{i\omega g(-1)} + \frac{1}{i\omega}\int_{-1}^{1} \frac{(f(x) - \mu[\sigma_0](x))}{g'(x)}(i\omega g'(x)e^{i\omega g(x)})\,dx \tag{4.15}$$

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx = \mu[\sigma_0](x)e^{i\omega g(x)}\Big|_{-1}^{1} + \frac{1}{i\omega}\left[\frac{(f(x) - \mu[\sigma_0](x))}{g'(x)}e^{i\omega g(x)}\right]\Bigg|_{-1}^{1} - \\ -\frac{1}{i\omega}\int_{-1}^{1} \frac{d}{dx}\left(\frac{(f(x) - \mu[\sigma_0](x))}{g'(x)}\right)e^{i\omega g(x)}\,dx \tag{4.16}$$

Now let $\sigma_1(x) := \frac{d}{dx}\left(\frac{(f(x)-\mu[\sigma_0](x))}{g'(x)}\right)$.

$$\int_{-1}^{1} f(x)e^{i\omega g(x)}\,dx = \mu[\sigma_0](x)e^{i\omega g(x)}\Big|_{-1}^{1} + \frac{1}{i\omega}\left[\frac{(f(x) - \mu[\sigma_0](x))}{g'(x)}e^{i\omega g(x)}\right]\Bigg|_{-1}^{1} - \\ -\frac{1}{i\omega}\int_{-1}^{1} \sigma_1(x)e^{i\omega g(x)}\,dx \tag{4.17}$$

And now applying the exact same procedure to the integral in (4.17) and inductively we construct the expansion. The pattern is identical to when we last did this in section (2.2). Let

$$\sigma_0(x) := f(x) \tag{4.18}$$

$$\sigma_{k+1}(x) := \frac{d}{dx}\left(\frac{\sigma_k(x) - \mathcal{L}[\mu[\sigma_k]](x)}{g'(x)}\right) \tag{4.19}$$

Then we have the expansion

$$I[f] \sim \sum_{k=0}^{\infty} \frac{1}{(i\omega)^k}\left(\mu[\sigma_k](x)e^{i\omega g(x)}\right)\Bigg|_{-1}^{1} + \sum_{k=0}^{\infty} \frac{1}{(i\omega)^{k+1}}\left(\frac{\sigma_k(x) - \mathcal{L}[\mu[\sigma_k]](x)}{g'(x)}\right)\Bigg|_{-1}^{1}. \tag{4.20}$$

And truncating to the first p terms gives us the moment-free asymptotic method,

$$Q_p^{\tilde{A}}[f,g] \equiv \sum_{k=0}^{p-1} \frac{1}{(i\omega)^k}\left(\mu[\sigma_k](x)e^{i\omega g(x)}\right)\Bigg|_{-1}^{1} + \sum_{k=0}^{p-1} \frac{1}{(i\omega)^{k+1}}\left(\frac{\sigma_k(x) - \mathcal{L}[\mu[\sigma_k]](x)}{g'(x)}\right)\Bigg|_{-1}^{1}. \tag{4.21}$$

Notice that we have placed a tilde on the A to separate it from the classic asymptotic method presented in section (2.2).

Next noting that

$$\mu[\sigma_k](\pm 1) = \mathcal{O}(\omega^{-1/r})$$

the asymptotic error of truncating the expansion readily follows as

$$I[f] - Q_p^{\tilde{A}}[f,g] \sim \omega^{-p-1/(r+1)} \quad \omega \to \infty.$$

4.1.1 Numerical Examples

Example 4.1.1 *Consider the integral*

$$\int_{-1}^{1} \cos(x) e^{i\omega(7x^2+x^3)}\, dx \qquad (4.22)$$

The $g(x)$ here has two stationary points but only one of them lies inside the interval $[-1,1]$ and has order 1. Another feature to emphasize here is that the second derivative is non-zero away from 0 in $[-1,1]$. So, we can use the moment free asymptotic method here.

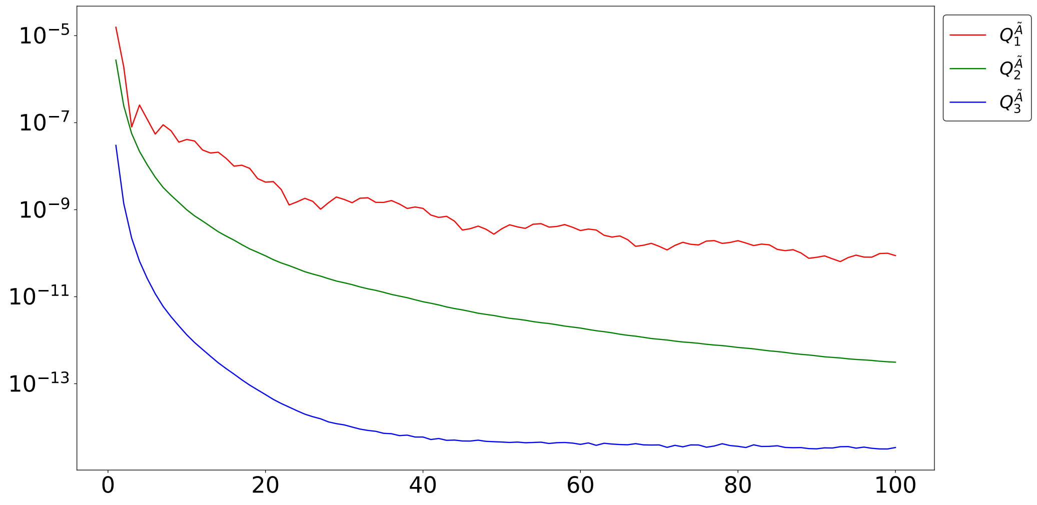


Figure 4.1: Log plot of the absolute error for the moment-free asymptotic method with 2,3 and 4 terms applied to integral (4.22).

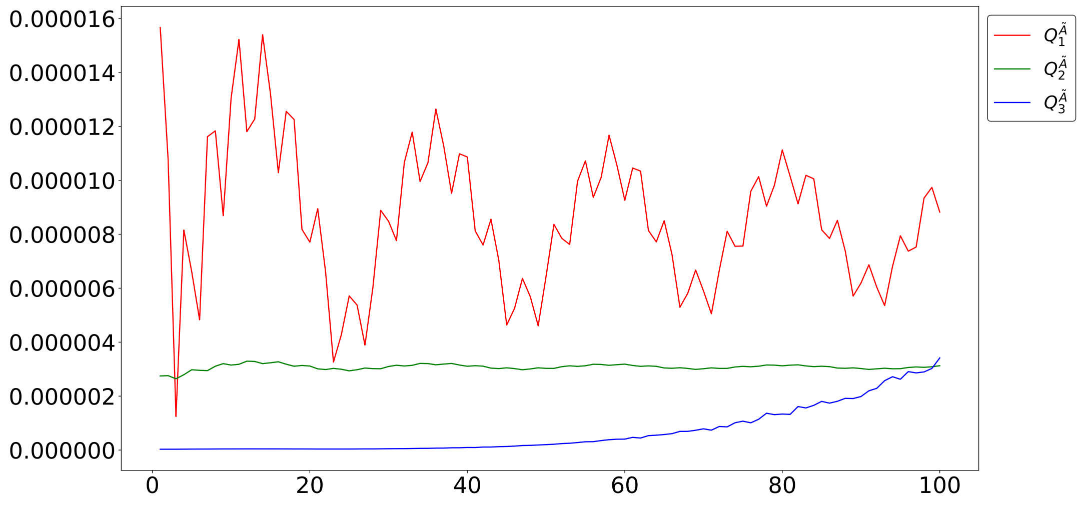


Figure 4.2: Absolute error scaled by $\omega^{+p+1/2}$ for the moment free asymptotic method with for $p = 1,2,3$ applied to (4.22).

Figures (4.1) and (4.2) show that the method evaluates the integral successfully. In Figure (4.2), we see that the error scaled by $\omega^{9/2}$ for $p = 3$ increases after a certain point, which is not something we would expect. This however is due to the fact that we fixed Maple's precision to 16 digits and as seen in Figure (4.1), $Q_3^{\tilde{A}}$ attains this.

Example 4.1.2 *Consider the integral*

$$\int_{-1}^{1} \frac{1}{1+x^2} e^{i\omega(1-\cos(x)-x^2/2+x^3)}\, dx \tag{4.23}$$

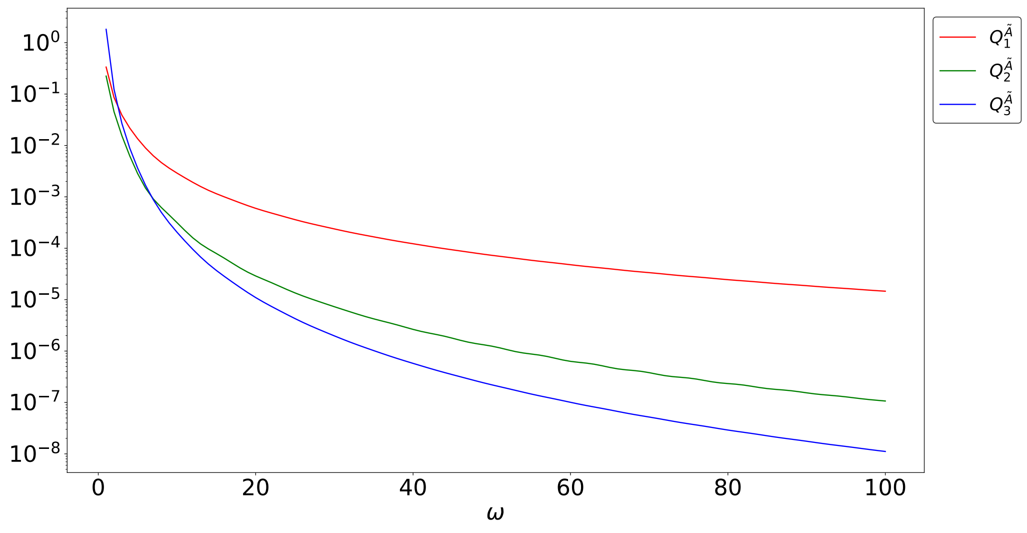


Figure 4.3: Log plot of the absolute error for the moment-free asymptotic method with 2,3 and 4 terms applied to integral (4.23).

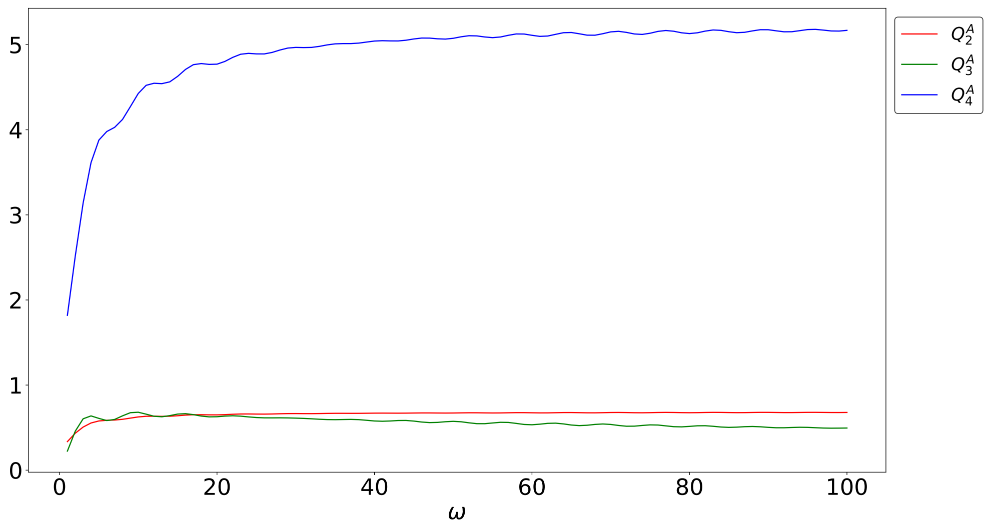


Figure 4.4: Absolute error scaled by $\omega^{p+1/3}$ for the moment free asymptotic method with for $p = 1, 2, 3$ applied to (4.23).

Both the plots show exactly what we would expect of the method. The result improves with additional terms and so does the rate of asymptotic error decay.

4.2 Moment-Free Filon Method

The moment-free Filon method is derived even more effortlessly than the moment-free asymptotic method. Instead of interpolating in the monomial basis, we simply interpolate in the new basis we derived earlier in this section.

Let $g(x)$ be a function with a single rth order stationary point. Let $\{\tau_k\}_{k=0}^{N-1}$ be $N \geq 3$ distinct points in $[-1, 1]$ such that $a = \tau_0 < \tau_1 < \ldots < \tau_\nu = \xi < \ldots < \tau_{N-1} = b$. With each interpolation point τ_i, associate a confluency $s_i \in \mathbb{N}\backslash\{0, 1\}$. So far we have the following two vectors,

$$\tau = [\tau_0, \tau_1, \ldots, \tau_{N-2}, \tau_{N-1}],$$
$$s = [s_0, s_1, \ldots, s_{N-2}, s_{N-1}].$$

Then let

$$\tilde{f}(x) = \sum_{k=0}^{d-1} c_k \mathcal{L}[\phi_{r,k}](x) \tag{4.24}$$

where $d = (-1 + \sum_{i=0}^{N-1} s_i)$. At each node τ_i with confluency s_i, we match our interpolating polynomial up to the $(s_i - 1)$th derivative of f at that point. That is,

$$\begin{aligned}
\tilde{f}(\tau_i) &= f(\tau_i)\\
\tilde{f}'(\tau_i) &= f'(\tau_i)\\
&\vdots\\
\tilde{f}^{(s_i-2)}(\tau_i) &= f^{(s_i-2)}(\tau_i)\\
\tilde{f}^{(s_i-1)}(\tau_i) &= f^{(s_i-1)}(\tau_i)
\end{aligned}$$

for $i = 0, 1, \ldots, N - 1$.

Then the Nth order Moment-free Filon method is given by

$$\begin{aligned}
Q_{N,s}^{\tilde{F}}[f, g] &\equiv \int_{-1}^{1} \tilde{f}(x) e^{i\omega g(x)}\, dx =\equiv \sum_{k=0}^{N-1} c_k I[\mathcal{L}[\phi_{r,k}]]\\
Q_{N,s}^{\tilde{F}}[f, g] &\equiv \sum_{k=0}^{d-1} c_k \left(\phi_{r,k}(1) e^{i\omega g(1)} - \phi_{r,k}(-1) e^{i\omega g(-1)}\right).
\end{aligned} \tag{4.25}$$

where $s = \min(s_0, s_{N-1})$ and $s_\nu = (2 \cdot s - 1) \cdot (r - 1)$ then the error decays as

$$I[f] - Q_{N,s}^{\tilde{F}}[f, g] \sim \omega^{-s-1/r} \quad \omega \to \infty. \tag{4.26}$$

4.2.1 Numerical Examples

Example 4.2.1 *Consider integrals of the form,*

$$\int_{-1}^{1} \cos(x) e^{i\omega x^r}\, dx \tag{4.27}$$

for $r \geq 2$. The function $g(x)$ has a stationary point of order r at $x = 0$. The exact expressions for the moments $\int_{-1}^{1} x^m e^{i\omega x^r}\, dx$ are known and we can use the classic Filon method to solve them (see Example 3.3.2).

Let's consider the case when $r = 4$. Then we need to sample at the stationary point node up to the $(2 \cdot s - 1)(3)$ derivatives. Figures (4.5) and (4.6) show the results.

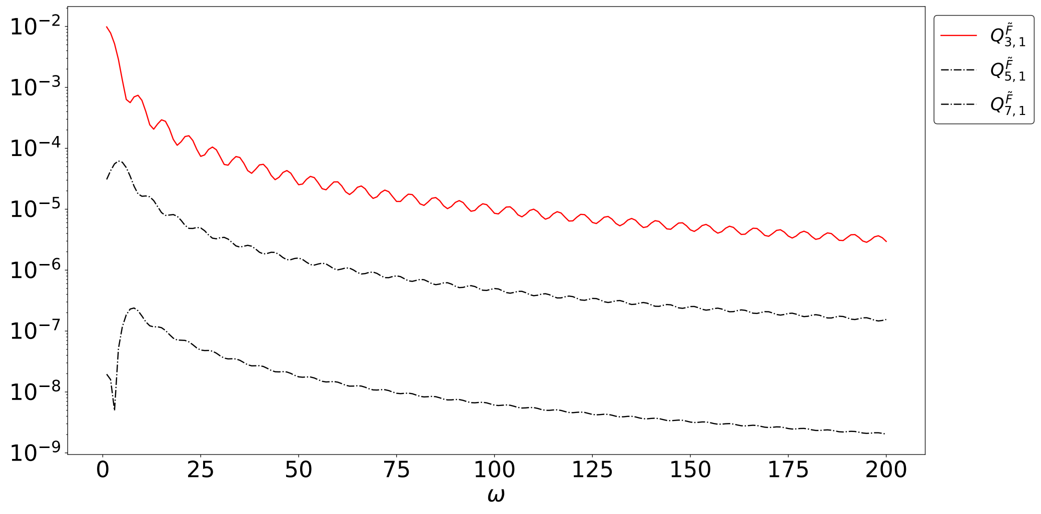


Figure 4.5: Absolute error on a log scale for number of nodes $N = 3, 5, 7$ with confluency 1 at the endpoints.

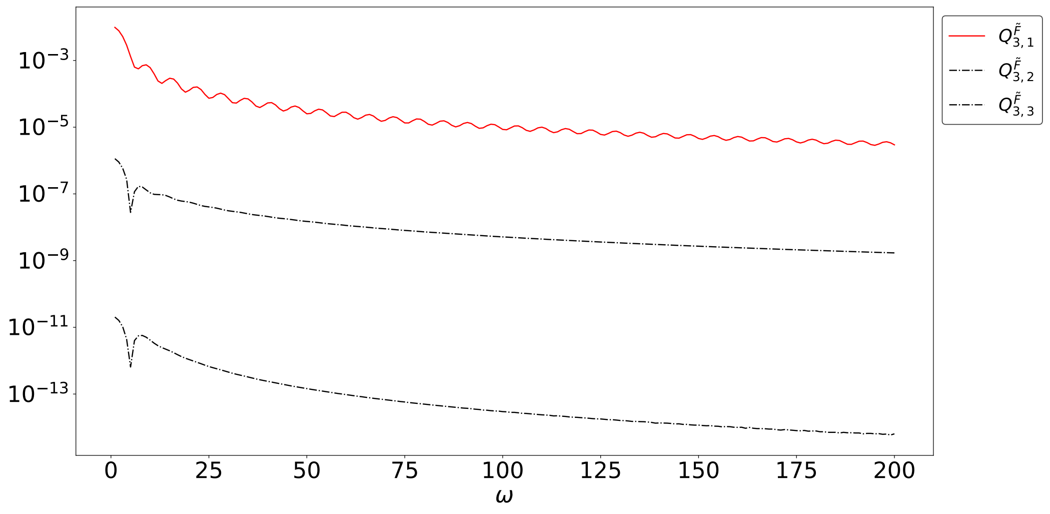


Figure 4.6: Absolute error on a log scale for 3 nodes with confluency $s = 1, 2, 3$ at the endpoints.

In Figure (4.5) we see that increasing the number of nodes decreases the magnitude of the absolute error. Figure (4.6) shows that the sampling derivatives of $f(x)$ at the endpoint has the same effect.

Example 4.2.2 *Consider the integral*

$$\int_{-1}^{1} \frac{1}{1+x^2} e^{i\omega(1-\cos(x)-x^2/2+x^3)}\, dx \tag{4.28}$$

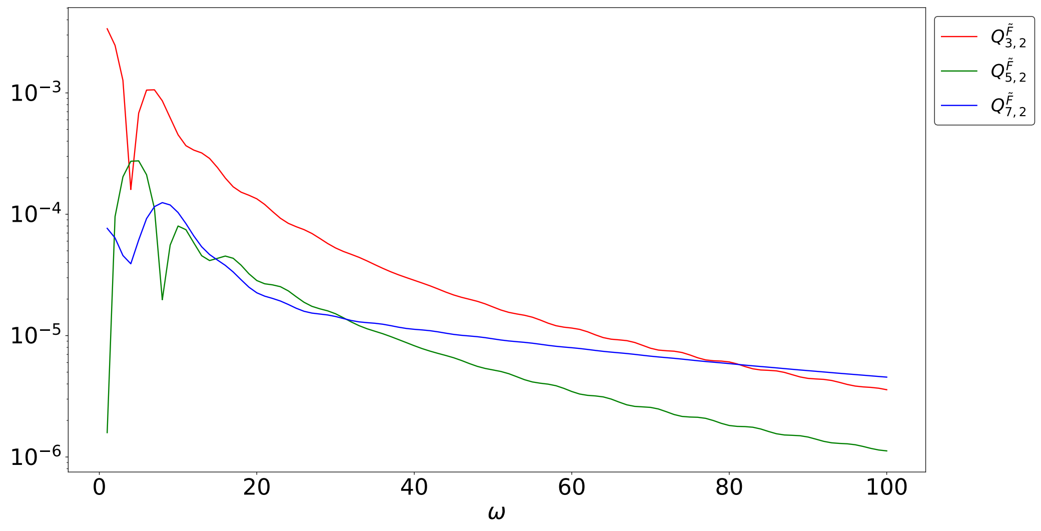


Figure 4.7: Log plot of the absolute error for the moment free Filon method applied to the integral (4.28) with the number of nodes $N = 3, 5, 7$.

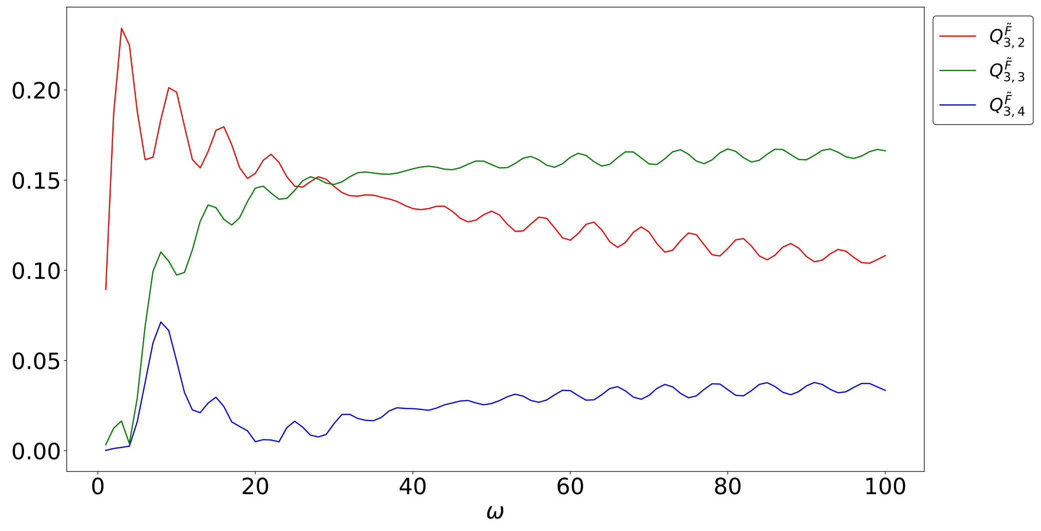


Figure 4.8: Absolute error scaled by $\omega^{s+1/3}$ for the moment free Filon method with 3 nodes, with confluencies $\{s, 2\cdot(2s-1), s\}$ for $s = 1, 2, 3$.

We conclude this chapter with a brief summary:

- Both the moment-free Filon and asymptotic method were derived by using the special basis constructed at the beginning of the chapter. This basis had the property that the expressions for the moments are always known.
- The basis depends on $g(x)$ and requires that $g''(x)$ not vanish anywhere besides 0 and $g^{(r+1)}(0) > 0$.
- Maple implementation for both these methods can be found in Appendix (A.4.2) and (A.4.3).

Chapter 5

Levin Type Methods

The idea behind Levin integration is to turn the problem of highly oscillatory quadrature into a problem of solving a polynomial ODE without boundary conditions. In general, numerically solving an ODE is much more difficult than quadrature but in this instance, because we restrict the solution to be polynomial, it makes the problem easier.

Consider the indefinite integral,

$$\int f(x)e^{i\omega g(x)}\,dx.$$

Assume that we have an antiderivative in closed form; namely that there is a function $P(x) \in C^1([a,b])$ such that the following equality holds

$$\int f(x)e^{i\omega g(x)}\,dx = P(x)e^{i\omega g(x)}.$$

Then the definite integral can be evaluated easily by the fundamental theorem of calculus as

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = P(b)e^{i\omega g(b)} - P(a)e^{i\omega g(a)}\,.s \tag{5.1}$$

This only works provided we can find such a $P(x)$. By changing the integrand if necessary, we will be able to do so.

We solve for $P(x)$ by differentiating both sides with respect to x,

$$\frac{d}{dx}\int f(x)e^{i\omega g(x)}\,dx = \frac{d}{dx}\left(P(x)e^{i\omega g(x)}\right)$$
$$f(x)e^{i\omega g(x)} = P'(x)e^{i\omega g(x)} + P(x)g'(x)e^{i\omega g(x)}$$

The exponential function $e^{i\omega g(x)} > 0$ for all $x \in \mathbb{R}$, so we have

$$f(x) = P'(x) + P(x)g'(x). \tag{5.2}$$

In effect, the problem of quadrature has been reduced to solving a first order ODE without boundary conditions. We solve this ODE using spectral methods that use differentiation matrices. This implicitly restricts $P(x)$ to be polynomial, and approximates our original $f(x)$ by another function (polynomial if $g'(x)$ is polynomial).

5.1 Levin-Hermite Quadrature

Given a set of $n+1$ Chebyshev-Lobatto nodes spanning the interval $[a, b]$,

$$\{\tau_k\}_{k=0}^{n} = \left\{ \frac{a+b}{2} + \frac{(b-a)}{2} \cos\left(\frac{\pi(n-k)}{n}\right) \right\}_{k=0}^{n} \in [a, b]$$

and values of the function $\{f(\tau_k)\}_{k=0}^{n}$, we can find a unique polynomial $p(z)$ of degree at most n, called the Lagrange interpolating polynomial, which has the property

$$p(\tau_i) = f(\tau_i), \quad i = 0, \cdots, n. \tag{5.3}$$

The first barycentric form (see [2] and [8]) of the Lagrange interpolating polynomial is given by

$$p(z) = w(z) \sum_{k=0}^{n} \frac{\beta_k \rho_k}{z - \tau_k},$$

where

$$w(z) = \prod_{i=0}^{n} (z - \tau_i),\ \beta_k = \frac{1}{\prod_{\substack{i=0 \\ i \neq k}}^{n} (\tau_k - \tau_i)} \text{ and } \rho_i = f(\tau_i).$$

If the Lagrange coefficients ρ_k are collected into a vector

$$\vec{\rho} = \begin{bmatrix} \rho_0 & \rho_1 & \cdots & \rho_n \end{bmatrix}^T,$$

we can construct a matrix $\mathbf{D_L}$, called the Lagrange differentiation matrix[1], such that

$$\vec{\rho'} = \mathbf{D_L} \vec{\rho},$$

where the vector $\vec{\rho'}$ are the Lagrange coefficients of the derivative of f, $f'(t)$.

The entries of $\mathbf{D_L}$ are given by

$$[\mathbf{D_L}]_{ij} = \begin{cases} \frac{-\beta_j}{\beta_i(\tau_i - \tau_j)} & i \neq j \\ -\sum_{\substack{k=0 \\ k \neq i}}^{n} [\mathbf{D_L}]_{ik} & i = j \end{cases}. \tag{5.4}$$

With the differentiation matrix at hand, the ODE (5.2) is approximately solved by solving the following linear system,

$$\vec{f} = \left(\mathbf{D_L} + i\omega \mathbf{I_{n+1}} \vec{g'}\right) \vec{P},$$

where

$$\vec{f} = \begin{bmatrix} f(\tau_0) & f(\tau_1) & \cdots & f(\tau_n) \end{bmatrix}^T,$$
$$\vec{g'} = \begin{bmatrix} g'(\tau_0) & g'(\tau_1) & \cdots & g'(\tau_n) \end{bmatrix}^T.$$

And $\mathbf{I_{n+1}}$ is an identity matrix of size $n+1$.

From which, the value of the integral is

[1]In [17], the authors call this the Chebyshev differentiation matrix.

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = P_n e^{i\omega g(\tau_n)} - P_0 e^{i\omega g(\tau_0)}, \tag{5.5}$$

where P_0 and P_n are the first and last entries of the solution vector $\vec{P}$.

This is the method described in [17]. However, instead of using Chebyshev polynomials, we have arrived at it using Lagrange interpolation (which can be performed on any set of discrete nodes, although some sets of nodes are better than others). The approach here can be extended to use Hermite interpolation, which allows us to use the information about the derivatives of f at the nodes.

Given a set of $n+1$ Chebyshev-Lobatto nodes $\{\tau_k\}_{k=0}^n$ spanning $[a,b]$, and at each τ_k, information of the function f upto the $(s_k-1)th$ derivative, we can then express a function $f(t)$ as

$$f(z) = w(z)\sum_{i=0}^{n}\sum_{j=0}^{s_i-1}\sum_{k=0}^{j}\beta_{i,j}\rho_{i,k}(z-\tau_i)^{k-j-1}$$

where $\beta_{i,j}$ are the barycentric weights and $\rho_{i,k}$ are the polynomial coefficients given by $\rho_{i,k} := \frac{f^{(k)}(\tau_i)}{k!}$. The barycentric weights $\beta_{i,j}$ are calculated by calculating the partial decomposition as follows,

$$\prod_{i=0}^{n}\frac{1}{(z-\tau_i)^{s_i}} = \sum_{i=0}^{n}\sum_{j=0}^{s_i-1}\frac{\beta_{i,j}}{(z-\tau_i)^{j+1}}. \tag{5.6}$$

As with Lagrange interpolation, if we put all of these coefficients $\rho_{i,k}$ into a vector of size $1+d$ ($d = -1 + \sum_{k=0}^{n} s_k$),

$$\begin{aligned}\vec{\rho} &= [\rho_{0,0}, \rho_{0,1}, \cdots, \rho_{0,s_0-1}, \cdots, \rho_{i,0}, \cdots, \rho_{i,s_i-1}, \cdots, \rho_{n,s_n-1}]^T \\ &= [\rho_1, \rho_2, \cdots, \rho_{s_i}, \cdots, \rho_{s_0+\cdots+s_i}, \cdots, \rho_{s_0+\cdots+s_i+s_i}, \cdots, \rho_{1+d}]^T,\end{aligned}$$

we can define a $(1+d)\times(1+d)$ matrix $\mathbf{D_H}$, called the Hermite differentiation matrix, such that

$$\vec{\rho'} = \mathbf{D_H}\vec{\rho}. \tag{5.7}$$

A derivation of the entries of $\mathbf{D_H}$ can be found in [4]. We have only presented the final result here.

In order to shorten notation, we define $s_{-1} = 0$ and our row and column numbering starts from 1. For each $k = 0, \cdots, n$, if $s_k > 1$, the set of rows enumerated by

$$k\text{th trivial rows set} = \{s_{-1} + s_0 + s_1 + \cdots + s_{k-1} + 1 + \eta\}_{\eta=0}^{s_k},$$

is a $(s_k - 1)\times d$ matrix,

$$\begin{bmatrix} \underbrace{0 \ \cdots \ 0}_{s_{-1}+s_0+\cdots+s_{k-1}+1} & 1 & 0 & & & & \cdots & 0 \\ & 0 & 2 & 0 & & & \cdots & 0 \\ \vdots & \vdots & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & (s_k-1) & 0 & \cdots & 0 \end{bmatrix}.$$

This leaves the non-trivial rows to be defined. These are the rows enumerated by the set

$$\text{Non-trivial rows} = \{s_0, s_0 + s_1, \cdots, s_0 + s_1 + \cdots + s_k, \cdots, 1 + d\}.$$

In order to write clean expressions for this, first we define

$$\beta_{i,j;k} = -\sum_{\mu=j}^{s_i-1} \beta_{i,\mu}(\tau_k - \tau_i)^{j-1-\mu},$$

where the $\beta_{i,j}$ on the right hand side are the generalized barycentric weights calculated by the partial fraction decomposition described in (5.6).

With this, the action of the k^{th} non-trivial row is given by

$$\begin{aligned}\frac{f^{(s_k)}(\tau_k)}{s_k!} =& \frac{1}{\beta_{k,s_k-1}}\left(-\sum_{j=0}^{s_k-2}\beta_{k,j}\frac{f^{(j+1)}(\tau_k)}{(j+1)!}+\right.\\ &-\left(\sum_{\substack{i=0\\i\neq k}}^{n}\sum_{j=0}^{s_i-1}\beta_{i,j}(\tau_k-\tau_i)^{-j-1}\right)f(\tau_k)\\ &\left.-\sum_{\substack{i=0\\i\neq k}}^{n}\sum_{j=0}^{s_i-1}\beta_{i,j;k}\frac{f^{(j)}(\tau_i)}{j!}\right).\end{aligned}$$

So the kth non-trivial row is given by

$$[\mathbf{D_H}]_{R_k,\nu} = \frac{1}{\beta_{k,s_k-1}} \cdot \begin{cases} \left(-\beta_{k,j}\right), & s_0 + \cdots + s_k + 1 < \nu < \\ & s_0 + \cdots + s_k + s_k, \\ -\left(\sum_{\substack{i=0\\i\neq k}}^{n} \sum_{j=0}^{s_i-1} \beta_{i,j}(\tau_k - \tau_i)^{-j-1}\right), & \nu = s_0 + \cdots + s_k \\ -\beta_{i,j;k}, & i = 0, \cdots, n; i \neq k \\ & j = 1, \cdots, s_i, \\ & \nu = s_{i-1} + j. \end{cases} .$$

where $R = \{s_0, s_0 + s_1, \cdots, s_0 + s_1 + \cdots + s_k, \cdots, 1 + d\}$. Note that the kth non-trivial row is not the kth row of the matrix, it is the R_kth row of the matrix.

Now with the matrix $\mathbf{D_H}$ defined, we can approximately solve the ODE (5.2) in the Hermite interpolational basis.

At each node τ_k, we need to solve for $\{P(\tau_k), P'(\tau_k), \frac{P''(\tau_k)}{2!}, \cdots, \frac{P^{(s_k-1)}(\tau_k)}{(s_k-1)!}\}$. To solve for the s_k unknowns at each node, we require the ODE (5.2) to be satisfied upto the $(s_k - 1)$th derivative,

$$\begin{aligned}
f(\tau_k) &= P'(\tau_k) + i\omega g'(\tau_k)P(\tau_k),\\
f'(\tau_k) &= P''(\tau_k) + i\omega(g''(\tau_k)P(\tau_k) + g(\tau_k)P'(\tau_k)),\\
&\vdots\\
f^{(j)}(\tau_k) &= P^{(j+1)}(\tau_k) + \sum_{\ell=0}^{j-1}\binom{j-1}{\ell}P^{(\ell)}(\tau_k)g^{(j-\ell)}(\tau_k), \qquad (5.8)\\
&\vdots\\
f^{(s_k-1)}(\tau_k) &= P^{(s_k)}(\tau_k) + \sum_{\ell=0}^{s_k-2}\binom{j-1}{\ell}P^{(\ell)}(\tau_k)g^{(s_k-2-\ell)}(\tau_k),
\end{aligned}$$

which gives up a total of $d+1$ equations for $d+1$ unknowns (s_k unknowns and equations at each τ_k and $d = -1 + \sum_{k=0}^{n} s_k$). In order to take the derivatives of P, we use the differentiation matrix $\mathbf{D_H}$. The construction of the system of equations is clear from the definition of $\mathbf{D_H}$ and the equations (5.8) at each node τ_k.

Upon solving these equations, the integral approximation is calculated by

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = P_{d-s_n+1}e^{i\omega g(b)} - P_1 e^{i\omega g(a)}. \qquad (5.9)$$

Note that if $s_k = 1$ for all k, then the Levin-Hermite method reduces to the Levin-Lagrange method.

Python code for generating the differentiation matrix and performing the integration will be made available through Github[2].

5.2 Levin-Bernstein Quadrature

The set of $n+1$ Bernstein polynomials of degree n,

$$\{B_k^n(x)\}_{k=0}^n = \left\{\binom{n}{k}x^k(1-x)^{n-k}\right\}_{k=0}^n,$$

forms a basis for polynomials of degree n over [0,1]. We define the modified Bernstein polynomials by,

$$\{\tilde{B}_k^n(x)\}_{k=0}^n = \left\{\binom{n}{k}\frac{(x-a)^k(b-x)^{n-k}}{(b-a)^n}\right\}_{k=0}^n,$$

which forms a basis for polynomials of degree n over $[a,b]$.

Because these constitute a polynomial basis, we can interpolate continuous functions in this basis. The coefficients are calculated by solving the following system:

[2] https://github.com/jeettrivedi/highly-oscillatory/

$$\begin{bmatrix} \tilde{B}_0^n(\tau_0) & \tilde{B}_1^n(\tau_0) & \cdots & \tilde{B}_n^n(\tau_0) \\ \tilde{B}_0^n(\tau_1) & \tilde{B}_1^n(\tau_1) & \cdots & \tilde{B}_n^n(\tau_1) \\ \vdots & & & \vdots \\ \tilde{B}_0^n(\tau_n) & \tilde{B}_1^n(\tau_n) & \cdots & \tilde{B}_n^n(\tau_n) \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} f(\tau_0) \\ f(\tau_1) \\ \vdots \\ f(\tau_n) \end{bmatrix}. \tag{5.10}$$

If we put the coefficients f_k into a vector $\vec{f} = \begin{bmatrix} f_0 & f_1 & \cdots & f_n \end{bmatrix}^T$, we can define an $(n+1) \times (n+1)$ matrix $\mathbf{D}_{\tilde{\mathbf{B}}}$, called the modified Bernstein differentiation matrix, which satisfies the following equality,

$$\vec{f'} = \mathbf{D}_{\tilde{\mathbf{B}}}\vec{f},$$

where $\vec{f'}$ are the coefficients which interpolate $f'(x)$ in the modified Bernstein basis.

The modified Bernstein differentiation matrix is a tridiagonal matrix. Its entries are as follows (see [1] for a proof):

$$[\mathbf{D}_{\tilde{\mathbf{B}}}]_{i,j} = \frac{1}{(b-a)} \cdot \begin{cases} 2i - n & i = j \\ -i & j = i - 1 \\ n - i & j = i + 1 \end{cases}. \tag{5.11}$$

The following is the Bernstein differentiation matrix over $[a, b]$ for 4 nodes,

$$[\mathbf{D}_{\tilde{\mathbf{B}}}]_{i,j} = \frac{1}{(b-a)} \cdot \begin{bmatrix} -3 & 3 & 0 & 0 \\ -1 & -1 & 2 & 0 \\ 0 & -2 & 1 & 1 \\ 0 & 0 & -3 & 3 \end{bmatrix}.$$

For two functions f and g expressed in the modified Bernstein basis of order n, their product fg can be expressed in the modified Bernstein basis of order $2n$. The coefficients of the product are given by

$$\{(fg)_i\}_{i=0}^{2n} = \left\{ \sum_{j=\max(0,i-n)}^{\min(i,2n)} \frac{\binom{n}{j}\binom{n}{i-j}}{\binom{2n}{i}} g_{i-j} f_j \right\}_{i=0}^{2n}. \tag{5.12}$$

Using the product formula (5.12) and the coefficients $\vec{f}$ of $f(x)$ in the modified Bernstein basis of order n, we define a $(2n+1) \times (n+1)$ matrix $\mathbf{M}_f$, which acts as follows

$$\overrightarrow{fg} = \mathbf{M}_f \vec{g}$$

where $\overrightarrow{fg}$ are the coefficients of the product $(fg)(x)$ in the modified Bernstein basis of order $2n$.

The entries of the banded matrix $\mathbf{M}_f$ are as follows:

$$\left[\mathbf{M}_f\right]_{i,j} = \begin{cases} \frac{\binom{n}{j}\binom{n}{i-j}}{\binom{2n}{i}} f_{i-j} & \max(0, i-n) \le j \le \min(i, 2n) \\ 0 & \text{otherwise} \end{cases}. \tag{5.13}$$

Using these matrices, we can solve the polynomial ODE (5.2) in the modified Bernstein basis by solving the following overdetermined system of linear equations,

$$\vec{f} = (\mathbf{M}_1 + i\omega\mathbf{M}_{g'})\vec{P}. \tag{5.14}$$

We solve this using Singular Value Decomposition (SVD) and find the solution with minimal 2-norm.

Lastly note that the modified Bernstein basis has the property,

$$\begin{aligned}\tilde{B}_k^n(a) &= \delta_{0,k},\\ \tilde{B}_k^n(b) &= \delta_{n,k}.\end{aligned}$$

This implied the first and last coefficients determine the value of solution at each of the end points respectively. With this, the value of the integral is given by

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = P_n e^{i\omega g(b)} - P_0 e^{i\omega g(a)}.$$

5.3 Levin-Compact Finite Difference Quadrature (Levin-CFD)

The idea of this method is very similar to that of Levin-Lagrange quadrature. Using the method of Compact-Finite differences, we define a matrix $\mathbf{D_C}$, which given a set of nodes and values of a function at these nodes, $\{\tau_k\}_{k=0}^n \in [a,b]$ and $\{f(\tau_k)\}_{k=0}^n$; can be used to calculate an approximation to the derivative of f at the nodes.

The derivation of such a matrix for the case of a uniform grid is covered in [16] and for the case of the non-uniform grid in [6, 4]. We simply present the matrix here for an arbitrary non-uniform grid $\{\tau_k\}_{k=0}^n$. Define $h_i \equiv \tau_{i+1} - \tau_i$. Then the matrix $\mathbf{D_C}$ is given by

$$\mathbf{D_C} = \mathbf{A}^{-1}\mathbf{B} \tag{5.15}$$

where the matrices $\mathbf{A}$ and $\mathbf{B}$ are defined as follows:

$$[\mathbf{A}]_{i,j} = \begin{cases} \frac{1}{(h_0+h_1)(h_0+h_1+h_2)} & i=0, j=0\\ \frac{1}{2(h_1)(h_1+h_2)} & i=0, j=1\\ \frac{1}{(h_i+h_{i-1})^2} & 0<i<n, j=i-1\\ \frac{1}{h_i^2} & 0<i<n, j=i\\ \frac{h_{i-1}^2}{(h_i+h_{i-1})^2 h_i^2} & 0<i<n, j=i+1\\ \frac{1}{h_{n-2}(h_{n-2}+h_{n-3})} & i=n, j=n-1\\ \frac{1}{(h_{n-1}+h_{n-2})(h_{n-1}+h_{n-2}+h_{n-3})} & i=n, j=n \end{cases}$$

$$[\mathbf{B}]_{i,j} = \begin{cases} \frac{4h_0^2+6h_0h_1+3h_0h_2+2h_1^2+2h_1h_2}{(h_0+h_1)^2(h_0+h_1+h_2)^2(h_0)} & i=0, j=0 \\ \frac{(-2h_1+h_0)h_2+2h_1(-h_1+h_0)}{h_0h_1^2(h_1+_2)^2} & i=0, j=1 \\ \frac{-h_0^2}{(h_0+h_1)^2h_1^2h_2} & i=0, j=2 \\ \frac{h_0^2}{(h_0+h_1+h_2)^2(h_1+h_2)^2h_2} & i=0, j=3 \\ \frac{4h_{i-1}+2h_i}{h_{i-1}(h_{i-1}+h_i)^3} & 0<i<n, j=i-1 \\ \frac{2(h_{i-1}-h_i)}{h_{i-1}h_i^3} & 0<i<n, j=i \\ -\frac{(4h_i+2h_{i-1})h_{i-1}^2}{(h_i+h_{i-1})^3h_i^3} & 0<i<n, j=i+1 \\ -\frac{h_{n-1}^2}{(h_{n-3}+h_{n-2}+h_{n-1})^2(h_{n-3}+h_{n-2})^2h_{n-3}} & i=n, j=n-3 \\ \frac{h_{n-1}^2}{(h_{n-2}+h_{n-1})^2h_{n-2}^2h_{n-3}} & i=n, j=n-2 \\ \frac{(2h_{n-2}-h_{n-1})h_{n-3}+2h_{n-2}(h_{n-2}+h_{n-1})}{h_{n-1}h_{n-2}^2(h_{n-3}+h_{n-2})^2} & i=n, j=n-1 \\ -\frac{4h_{n-1}^2+h_{n-1}(6h_{n-2}+3h_{n-3})+2h_{n-2}(h_{n-3}+h_{n-2})}{h_{n-1}(h_{n-2}+h_{n-1})^2(h_{n-3}+h_{n-2}+h_{n-1})^2} & i=n, j=n \end{cases}$$

Using the matrix $\mathbf{D_C}$, ODE (5.2) is approximately solved by solving the following system of equations:

$$\vec{f} = \left(\mathbf{D_C} + i\omega\mathbf{I}\vec{g'}\right)\vec{P}.$$

Matrix inversion is computationally costly so instead of explicitly calculating $\mathbf{D_C}$, we solve the following equivalent system,

$$\mathbf{A}\vec{f} = \left(\mathbf{B} + i\omega\mathbf{A}\vec{g'}\right)\vec{P} \tag{5.16}$$

From the solution of (5.16), the integral is calculated by

$$\int_a^b f(x)e^{i\omega g(x)}\,dx = P_n e^{i\omega g(b)} - P_0 e^{i\omega g(a)}$$

5.4 Numerical Experiments

Example $I_1 = \int_{-1}^{1}(x^3 + x^2 + x)e^{i\omega x}\, dx$

This integral can be calculated exactly using integration by parts,

$$[I_1]_{\text{exact}} = -\frac{e^{-i\omega}}{\omega^4}(3i\omega^3 e^{2i\omega} - 8i\omega e^{2i\omega} - 6\omega^2 e^{2i\omega} - 4i\omega + 2\omega^2 - 6).$$

f is a 3rd order polynomial, so we expect the Levin-Lagrange and Levin-Bernstein to be exact with 4 nodes and Levin-Hermite with 2 nodes of confluency 2 each. For the Levin-CFD method, we also expect the result to be exact with 5 nodes.

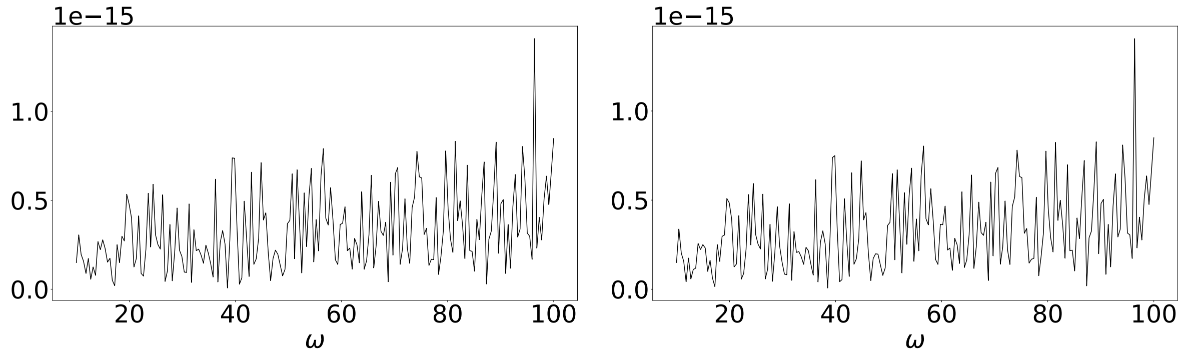


Figure 5.1: Absolute error using the Levin-Lagrange method (Left) with 4 Chebyshev nodes and Levin-Hermite method with 2 nodes each with confluency 2 (Right).

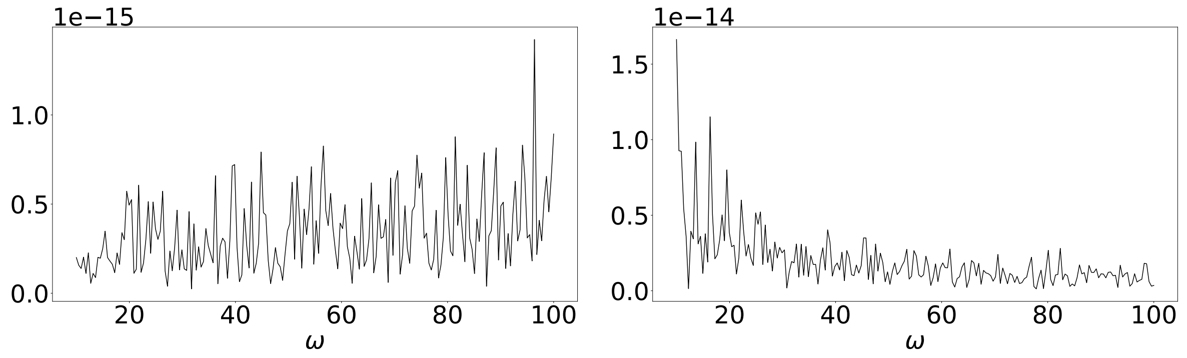


Figure 5.2: Absolute error using the Levin-Bernstein method (Left) with 4 Chebyshev nodes and Levin-CFD method with 5 nodes (Right).

As seen in Figure 5.1 and 5.2, the results are exact upto machine epsilon, as expected.

The next three examples demonstrate the asymptotic order of each of the method as $\omega \to \infty$. All four methods retain the property that the accuracy increases as ω increases.

Example $I_2 = \int_{-1}^{1} (\frac{1}{1+x^2}) e^{i\omega x}\, dx$

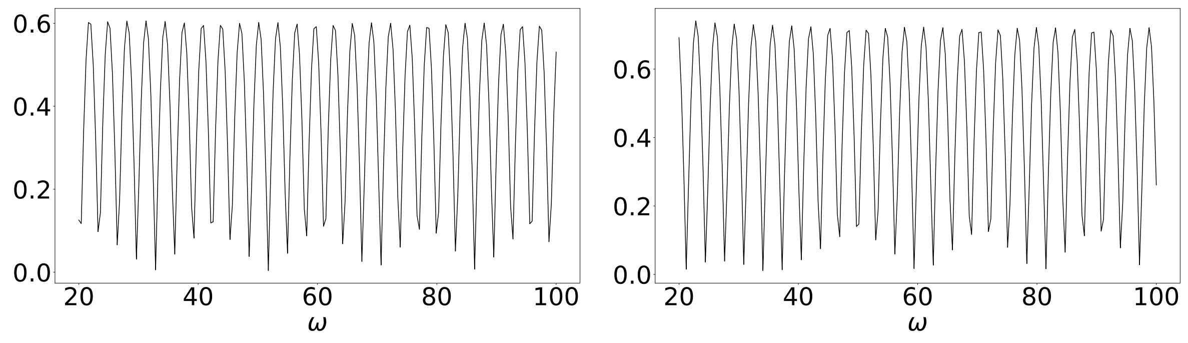


Figure 5.3: Absolute error multiplied by ω^2 using the Levin-Lagrange method (Left) with 4 Chebyshev nodes and Levin-Hermite method with 4 nodes each with confluency 2 multiplied by ω^3 (Right).

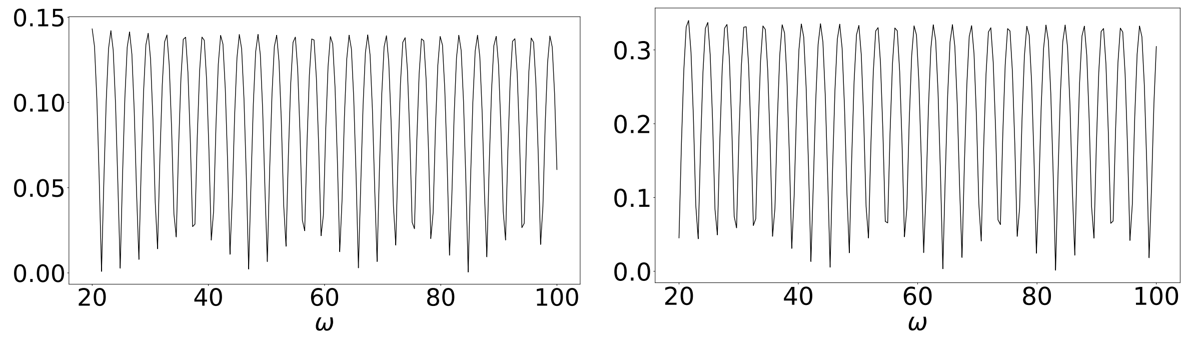


Figure 5.4: Absolute error multiplied by ω using the Levin-Bernstein method (Left) with 4 Chebyshev nodes and Levin-Composite Finite Difference method with 5 nodes multiplied by ω^2 (Right).

Example $I_3 = \int_1^2 (\frac{1}{1+x^2}) e^{i\omega x^2}\, dx$

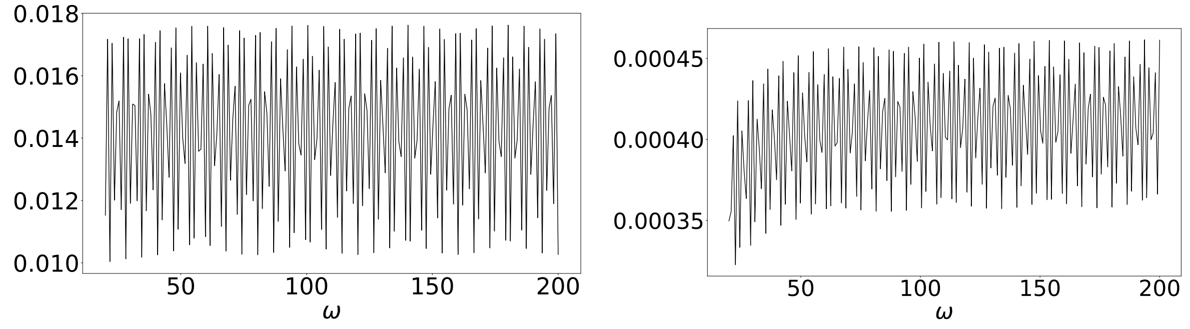


Figure 5.5: Absolute error multiplied by ω^2 using the Levin-Lagrange method (Left) with 4 Chebyshev nodes and Levin-Hermite method with 4 nodes each with confluency 2 multiplied by ω^3 (Right).

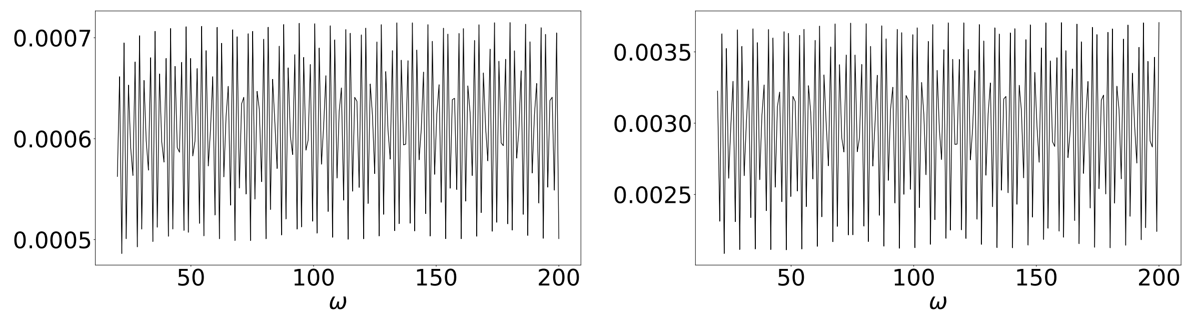


Figure 5.6: Absolute error multiplied by ω using the Levin-Bernstein method (Left) with 3 Chebyshev nodes and Levin-CFD method with 5 nodes multiplied by ω^2 (Right).

Example $I_4 = \int_1^2 (e^x) e^{i\omega cosh(x)}\, dx$

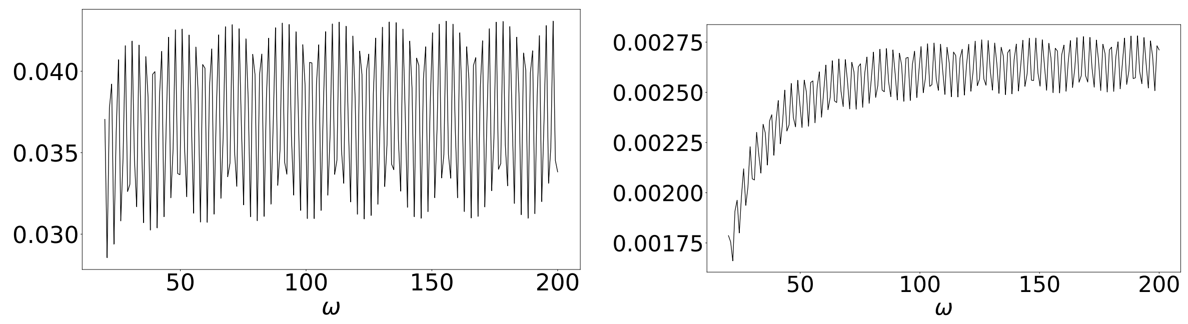


Figure 5.7: Absolute error multiplied by ω^2 using the Levin-Lagrange method (Left) with 4 Chebyshev nodes and Levin-Hermite method with 4 nodes each with confluency 2 multiplied by ω^3 (Right).

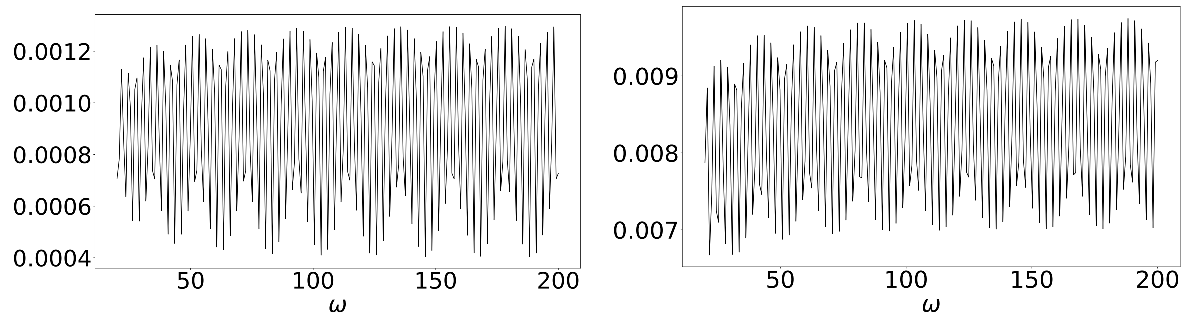


Figure 5.8: Absolute error multiplied by ω using the Levin-Bernstein method (Left) with 3 Chebyshev nodes and Levin-Composite Finite Difference method with 4 nodes multiplied by ω^2 (Right).

Example $I_5 = \int_{-1}^1 \int_{-1}^1 f(x,y) e^{i\omega g(x,y)}\, dx$

The Levin-Hermite method and the Levin-CFD method can be used to evaluate multiple integrals over rectangular domains using the method of delaminating quadrature. This approach for Levin integration was proposed in [18]. The extension of Levin-Hermite and Levin-CFD to do multiple integrals of this form is straightforward. We present here the case of Levin-Hermite with confluency 2 at each node.

$$I_5 = \int_{-1}^{1} \int_{-1}^{1} f(x,y) e^{i\omega g(x,y)}\, dx, \tag{5.17}$$

where $g(x,y), f(x,y)$ are continuous.

First calculate the inner integral of (5.17),

$$\int_{-1}^{-1} f(x,y) e^{i\omega g(x,y)}\, dy,$$

using 1 dimensional Levin integration method, which requires us to solve the PDE

$$f = p_y + i\omega g_y p,$$

where the subscripts denote partial derivative with respect to that variable.

This leads to

$$\int_{-1}^{-1} f(x,y)e^{i\omega g(x,y)}\, dy = p(x,1)e^{i\omega g(x,1)} - p(x,-1)e^{i\omega g(x,-1)}. \tag{5.18}$$

So, the complete integral can be calculated by substituting the right hand side of (5.18) in place of the inner integral in (5.17),

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)e^{i\omega g(x,y)}\, dy\, dx = \int_{-1}^{1} p(x,1)e^{i\omega g(x,1)} - p(x,-1)e^{i\omega g(x,-1)}\, dx,$$

which is two 1 dimensional integrals, which we can solve, once we have found $p(x,y)$.

Construct a tensor grid of Chebyshev-Lobatto nodes that spans the domain with N^2 nodes. Label them

$$\{(x_j, y_k)\}_{j,k=0}^{N-1}.$$

For each $j = 0, \cdots, N-1$, we have to solve the following $2N$ equations,

$$p_y(x_j, y_m) + i\omega g_y(x_j, y_m)p(x_j, y_m) = f(x_j, y_m), \quad m = 0, \cdots, N-1$$

and

$$p_{yy}(x_j, y_m) + i\omega(g_{yy}(x_j, y_m)p(x_j, y_m) + g_y(x_j, y_m)p_y(x_j, y_m)) = f_y(x_j, y_m),$$
$$m = 0, \cdots, N-1.$$

The complete set of solutions gives us the values of $p(x,y)$ and $p_y(x,y)$ at the grid points.

Now we are in the position to solve the two integrals, which we do as follows.

$$\int p(x,1)e^{i\omega g(x,1)} = Q_1(x)e^{i\omega g(x,1)}$$

where $Q_1(x)$ is sufficiently continuously differentiable. Now, we solve the associated ODE

$$\frac{dQ_1(x)}{dx} + \omega\frac{g(x,1)}{dx}Q_1(x) = p(x,1)$$

by solving the following N equations,

$$(Q_1)_x(x_k,1) + i\omega g_x(x_k,1)(Q_1)(x_k,1) = f(x_k,1) \quad k = 0, \cdots, N-1$$

which gives us the values of $Q_1(x)$, with which we calculate the integral. And so the above integral has the value

$$\int_{-1}^{1} p(x,1)e^{i\omega g(x,1)} = Q_1(1)e^{i\omega g(1,1)} - Q_1(-1)e^{i\omega g(-1,1)}$$

Similarly we solve the second integral,

$$\int_{-1}^{1} p(x,-1)e^{i\omega g(x,-1)} = Q_2(1)e^{i\omega g(1,-1)} - Q_2(-1)e^{i\omega g(-1,-1)}$$

And thus the complete integral is given by

$$\begin{aligned}\int_{-1}^{1}\int_{-1}^{1} f(x,y)e^{i\omega g(x,y)}\, dy\, dx =& Q_1(1)e^{i\omega g(1,1)} - Q_1(-1)e^{i\omega g(-1,1)} \\ &- (Q_2(1)e^{i\omega g(1,-1)} - Q_2(-1)e^{i\omega g(-1,-1)})\end{aligned}$$

We demonstrate this with the following intergral,

$$I_5 = \int_{-1}^{1}\int_{-1}^{1} \cos(x+y)e^{i\omega(x+y)}\, dy\, dx. \tag{5.19}$$

The exact value of this integral is given by

$$\begin{aligned}[I_5]_{\text{exact}} =& -\frac{1}{\omega^4 - 2\omega^2 + 1}\Big(e^{4i\omega}\cos(2)\omega^2 - 2ie^{4i\omega}\sin(2)\omega \\ &+e^{4i\omega}\cos(2) - 2e^{2i\omega}\omega^2 + 2i\omega\sin(2) - 2e^{2i\omega} + \cos(2)\Big)\end{aligned} \tag{5.20}$$

When we evaluate the outer integral, we are only using the information of $p(x,y)$ and not its derivative, so we expect the error to decay as $O(\omega^{-2})$.

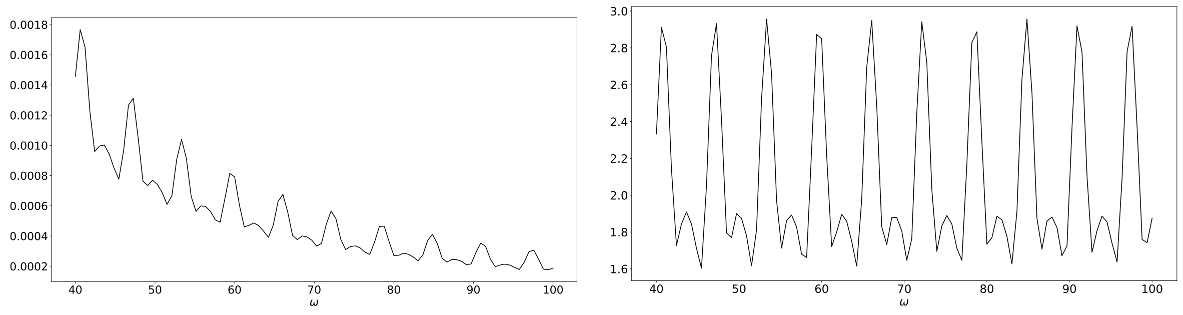


Figure 5.9: Absolute error using the Levin-Hermite method (Left) with 5 Chebyshev nodes, each with confluency 2 and the absolute error of the results scaled by ω^2 (Right).

As seen in the figure, we get 4 digits of accuracy with 5 nodes and the absolute error decays as $O(\omega^{-2})$.

5.4.1 Stability of Methods

Next, we demonstrate the stability of the methods with two examples. For both, we fix ω to 200 and raise the number of nodes used.

Example $I_6 = \int_1^2 (\frac{1}{1+x^2}) e^{200ix^2}\, dx$

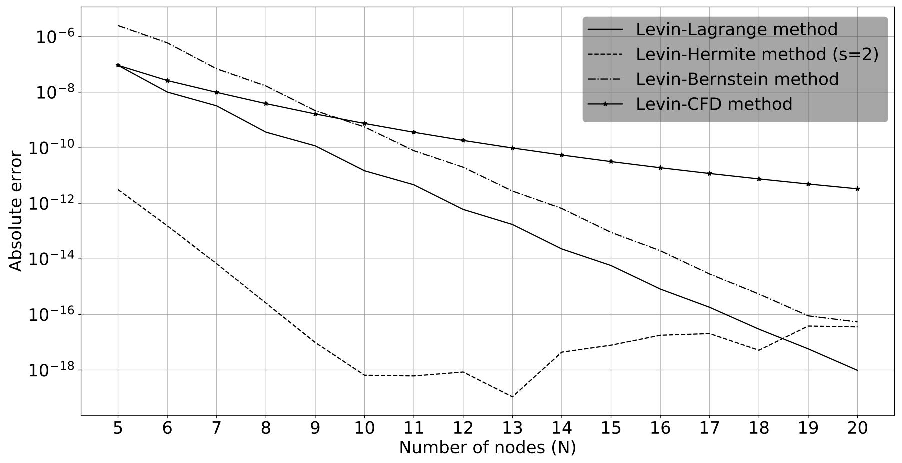


Figure 5.10: Log plot of the error as a function of the number of nodes for each of the methods starting from 5 nodes and going up to 20 nodes.

Example $I_7 = \int_1^2 (\frac{1}{1+x^2}) e^{200icosh(x)}\, dx$

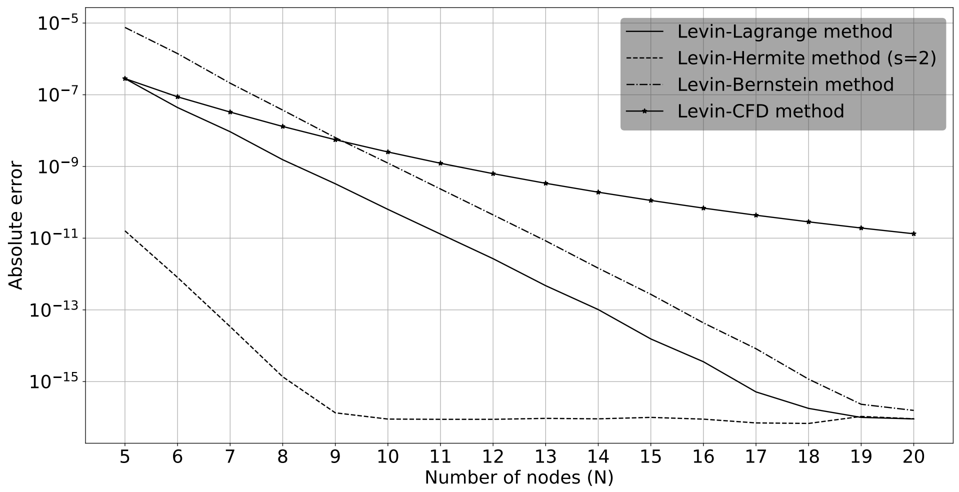


Figure 5.11: Log plot of the error as a function of the number of nodes for each of the methods for I_7 starting from 5 nodes and going up to 20 nodes.

Lastly we show that for the Levin-Hermite method, the method is stable with respect to raising of the confluency at the nodes.

Example $I_4 = \int_1^2 (e^x) e^{200icosh(x)}\, dx$

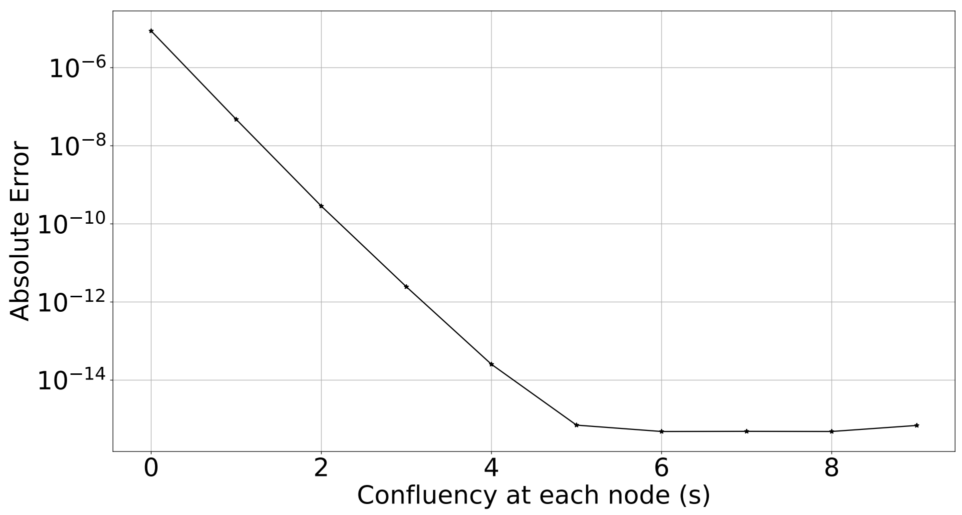


Figure 5.12: Log plot of the absolute error for Levin-Hermite method as a function of the confluency at the 2 nodes (one at each endpoint) for I_4.

Chapter 6

Hybrid Method & Concluding Remarks

Each of the methods described in the chapters before either works only in the absence of stationary points or requires there to be only one stationary point in the integration interval. Furthermore, the moment free methods have additional constraints which have to be taken into account before it can be used.

Apart from the constraints, the methods require additional information such as the stationary points and their orders. The idea of the hybrid method presented here is to abstract away this complexity from the user and allows for easy inclusion into existing integration packages.

The details of the method are fairly straightforward and don't require a great deal of elaboration. We sum up the essential idea of the method in the steps below:

1. Find all the stationary points by solving $g'(x) = 0$ and keep only the ones that lie in $[a, b]$.
2. For each stationary points, evaluate further derivatives of $g(x)$ at that point until a non-zero derivative is reached. This gives us the order of each stationary point.
3. Find all the points in $[a, b]$ where the second derivative vanishes but the first derivative doesn't.
4. For each stationary point, make a list of all the problematic points that lie to the left of it and to the right of it. Using this list, we can extract an interval centered around the stationary points that excludes all of these points. Over each such interval, we need to use a method that can handle stationary points.
    - If $g^{(r+1)}$(stationary point) > 0, use moment-free Filon method on this interval.
    - If $g^{(r+1)}$(stationary point) < 0, use the regular Filon method on this interval.
5. Lastly check what parts of the interval are not covered by the previous step and apply Levin-Hermite method on these intervals.

Using these ideas, it is easy enough to write a an implementation in the computer algebra system of your choice. We have included a Maple implementation in Appendix (A.5).

This brings us to the last part of this thesis, the concluding remarks. It is worth noting that at the time of writing, neither Maple nor Mathematica have special methods implemented for highly oscillatory integrals of the form discussed in this thesis. The code included in this thesis along with the hybrid method is a first step in this direction.

That being said, the Maple prototype presented in Appendix (A.5) is a very naive implementation of the idea and is a long way from what one would expect of a final product. We conclude this work with suggestions for further work in the direction of improving this code.

An obvious improvement would be to add some form of error control in the method. A good starting point for this is the paper [12] which provides error bounds for both the Filon and the Asymptotic method. There is an even simpler approach to error control when calculating integrals for a large range of ω. For the lowest ω value in the range, compute the absolute error. We know the rate at which the error decays with ω and so using this information, we can adjust our parameters accordingly to reach the desired tolerance.

An equally important improvement would be generalizing the moment-free basis discussed in chapter (4) to include the case where $g^{(r+1)}$(stationary point) < 0. In the original paper [21], the author states that this condition can be relaxed at the expense of complicating proofs. This would indicate that such a generalization is trivial but all attempts so far appear to indicate otherwise.

Besides this, additional methods such as analytic continuation and steepest descent (described in [9]) can be incorporated into this method as well. Other directions one could take is to consider other oscillatory kernels such as the Bessel kernel (see [3],[23]) and the more general class of integrals explored in [24] and extend the method to account for those. Implementations for methods for multivariate highly oscillatory integrals (see [11],[14]) can also be added.

There is sufficient reliable literature on the topic of numerical quadrature of highly oscillatory integrals to warrant the creation of an integration package that will be able to solve a large number of problems arising in applications. This thesis is a first step in the direction of this goal.

Bibliography

[1] Amirhossein Amiraslani, Robert M. Corless, and Madhusoodan Gunasingham. Differentiation matrices for univariate polynomials. *submitted*, 2018.

[2] Jean-Paul Berrut and Lloyd N Trefethen. Barycentric Lagrange interpolation. *SIAM review*, 46(3):501–517, 2004.

[3] Ruyun Chen. Numerical approximations to integrals with a highly oscillatory bessel kernel. *Applied Numerical Mathematics*, 62(5):636–648, 2012.

[4] Robert M Corless and Nicolas Fillion. A graduate introduction to numerical methods. *AMC*, 10:12, 2013.

[5] Louis Napoleon George Filon. Iii.—on a quadrature formula for trigonometric integrals. *Proceedings of the Royal Society of Edinburgh*, 49:38–47, 1930.

[6] L Gamet, F Ducros, Franck Nicoud, and Thierry Poinsot. Compact finite difference schemes on non-uniform meshes. application to direct numerical simulations of compressible flows. *International Journal for Numerical Methods in Fluids*, 29(2):159–191, 1999.

[7] Richard M Goody and Yuk Ling Yung. *Atmospheric radiation: theoretical basis*. Oxford university press, 1995.

[8] Nicholas J Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24(4):547–556, 2004.

[9] Daan Huybrechs and Stefan Vandewalle. On the evaluation of highly oscillatory integrals by analytic continuation. *SIAM Journal on Numerical Analysis*, 44(3):1026–1048, 2006.

[10] A Iserles, SP Nørsett, and S Olver. Highly oscillatory quadrature: The story so far. In *Numerical mathematics and advanced applications*, pages 97–118. Springer, 2006.

[11] Arieh Iserles and Syvert Nørsett. Quadrature methods for multivariate highly oscillatory integrals using derivatives. *Mathematics of computation*, 75(255):1233–1258, 2006.

[12] Arieh Iserles and Syvert P Nørsett. On quadrature methods for highly oscillatory integrals and their implementation. *BIT Numerical Mathematics*, 44(4):755–772, 2004.

[13] Arieh Iserles and Syvert P Nørsett. Efficient quadrature of highly oscillatory integrals using derivatives. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 461, pages 1383–1399. The Royal Society, 2005.

[14] Arieh Iserles and Syvert P Nørsett. From high oscillation to rapid approximation iii: Multivariate expansions. *IMA journal of numerical analysis*, 29(4):882–916, 2009.

[15] John David Jackson. Classical electrodynamics, 1999.

[16] Sanjiva K Lele. Compact finite difference schemes with spectral-like resolution. *Journal of computational physics*, 103(1):16–42, 1992.

[17] JianBing Li, Xuesong Wang, and Tao Wang. A universal solution to one-dimensional oscillatory integrals. *Science in China Series F: Information Sciences*, 51(10):1614–1622, 2008.

[18] Jianbing Li, Xuesong Wang, Tao Wang, and Chun Shen. Delaminating quadrature method for multi-dimensional highly oscillatory integrals. *Applied Mathematics and Computation*, 209(2):327–338, 2009.

[19] FWJ Olver. Error bounds for stationary phase approximations. *SIAM Journal on Mathematical Analysis*, 5(1):19–29, 1974.

[20] Sheehan Olver. Moment-free numerical integration of highly oscillatory functions. *IMA Journal of Numerical Analysis*, 26(2):213–227, 2006.

[21] Sheehan Olver. Moment-free numerical approximation of highly oscillatory integrals with stationary points. *European Journal of Applied Mathematics*, 18(4):435–447, 2007.

[22] EM Stein. Harmonic analysis: Real variable methods, orthogonality and oscillatory integrals vol. 43 of the princeton math. *Series. Princeton U. Press. Princeton, NJ*, 1993.

[23] Shuhuang Xiang, Yeol Je Cho, Haiyong Wang, and Hermann Brunner. Clenshaw–curtis–filon-type methods for highly oscillatory bessel transforms and applications. *IMA Journal of Numerical Analysis*, 31(4):1281–1314, 2011.

[24] Shuhuang Xiang and Haiyong Wang. Fast integration of highly oscillatory integrals with exotic oscillators. *Mathematics of Computation*, 79(270):829–844, 2010.

Appendix A

Computer Code

All of the code in the Appendix can be downloaded for use from Github using the following link: `https://github.com/jeettrivedi/highly-oscillatory/`.

A.1 Asymptotic-Type Methods

A.1.1 Asymptotic Method in the absence of stationary points

```
Asymptotic_Method_no_stat:=proc(f_t,g_t,a,b,p,omega_range::list)

  description "Computes Highly Oscillatory integrals with an rth order stationary point in the interval";
  local Q_A, f, g, h, k, mu, omega, rho, vals, xi;

  # Transforming the integral
  f:=x->f_t(x*(b-a)+a);
  g:=x->g_t(x*(b-a)+a);

  # Calculating the coefficients
  rho[0]:= f(x);
  for k from 1 to p-1 do
    rho[k]:= diff(rho[k-1]/diff(g(x),x),x);
  od;

  # Calculating the integral values
  return [ seq(evalf(eval(-(b-a)*add(1/(-I*w)^(m+1)*(eval(exp(I*w*g(x))*rho[m]/diff(g(x),x),x=1)-eval(exp(I*w*g(x))*rho[m]/diff(g(x),x),x=0)),m=0..p-1),w=omega)),omega = omega_range) ]:

end proc;
```

A.1.2 Asymptotic Method in the presence of stationary points

```
Asymptotic_Method_stat_points:=proc(f_t,g_t,a,b,xi_t,r,p,
   omega_range::list)

  # description "Computes Highly Oscillatory integrals of the
     form
  # int(f(x)*exp(I*w*g(x)),x=a..b)
  # with an rth order stationary point in the interval and w
     in omega_range":

  local f,g,xi,mu,rho,k,h,vals,omega:

  Digits := 16:

  # Transforming the integral
  f:=x->f_t(x*(b-a)+a);
  g:=x->g_t(x*(b-a)+a);
  xi:=(xi_t-a)/(b-a);

  # The ‘‘simpler’’ highly oscillatory integrals
  mu := (n, omega, xi)->int((x-xi)^n*exp(I*omega*g(x)), x =
     0..1);

  # Calculating the coefficients
  rho[0]:= F(x);
  for k from 1 to p-1 do
    rho[k]:= Diff((rho[k-1]-add((Limit((Diff(rho[k-1], [x$j]))
       /factorial(j), x = xi))*(x-xi)^j, j = 0 .. r-1))/(Diff(
       G(x), x)), x);
  od:

  for k from 0 to p-1 do
    rho[k] := simplify(value(eval(rho[k], {F(x) = f(x), G(x) =
        g(x)})));
  od:

  return [ seq( evalf(value(add(evalf(mu(j, omega, xi))*add((
     Limit(value(diff(rho[m], [x$j])), x = xi))/(-I*omega)^m,
     m = 0 .. p-1)/factorial(j), j = 0 .. r-1))-add((exp(I*
     omega*g(1))*(map(w->eval(w, x = 1),rho)[m]-map(w->limit(w
     , x = xi), rho)[m])/(eval(diff(g(x), x), x = 1))-exp(I*
     omega*g(0))*(map(w->eval(w, x = 0),rho)[m]-map(w->limit(w
     , x = xi),rho)[m])/(eval(diff(g(x),x), x=0)))/(-I*omega)
     ^(m+1), m = 0 .. p-1))*(b-a), omega=omega_range) ]:
```

```


end proc;
```

A.2 Filon-Type Methods

A.2.1 Filon-Lagrange Method

```
Filon_Method_No_Stat:=proc(f_t::algebraic,g_t::algebraic,a::
    numeric,b::numeric,N::integer,omega_range::list)
  description "Computes highly oscillatory integrals of the
      form
  Int(f(x)*exp(I*w*g(x)),x=a..b).
  using Filon-Lagrange integration. The method requires that g
      (x) have no stationary points in the interval [a,b].";


  local Int_f, array_of_funcs, f, f_tau, g, i, j, p, tau, vals
      , wts, wts_pre_calc;
  Digits:=16:

  # Transforming the integral
  f := x->f_t(x*(b-a)+a);
  g := x->g_t(x*(b-a)+a);

  # Generating interpolation points
  tau := [seq((1+cos(Pi*i/(N-1)))*(1/2), i = N-1 .. 0, -1)];
  f_tau := [seq(f(tau[i+1]), i = 0 .. N-1)];

  # Calculating the moments
  wts := [seq(int(x^i*exp(I*omega*g(x)), x = 0 .. 1), i = 0 ..
       N-1)];

  # Interpolation
  p := z->CurveFitting[PolynomialInterpolation](zip(`[]`, tau,
       f_tau), z);
  p := collect(p(z), z);
  Int_f := (b-a)*((subs(z = 0, p)*wts[1]+add(coeff(p, z^(i-1))
      *wts[i], i = 2 .. N)));

  # Calculation of the integral
  return [ seq(evalf(subs(omega = w, Int_f)),w=omega_range) ]:

end proc:
```

A.2.2 Filon-Hermite Method

```
Filon_Method_Stat_Pts:=proc(f_t::algebraic,g_t::algebraic,a,b,
    N::integer,s_t,xi_t,r::integer,omega_range::list)
  description "Computes highly oscillatory integrals of the
```

```
   form
Int(f(x)*exp(I*w*g(x)),x=a..b).
using Filon-Hermite integration. The method requires that g(
   x) have only 1 stationary point in [a,b]";

local appended_zero,Int_f, array_of_funcs, f, f_tau, g, i,
   interpolant_coeffs, j, p, s, tau, vals, wts, wts_pre_calc
   , xi, xy;

Digits:=16:
# Transforming the integral
f:=x->(b-a)*f_t(x*(b-a)+a);
g:=x->g_t(x*(b-a)+a);
xi:=(xi_t-a)/(b-a);

# Generating interpolation points
tau := Array([seq((1+cos(Pi*i/(N-1)))*(1/2), i = N-1 .. 0,
   -1)]);

# Adding stat point as a node if it is not a node already
appended_zero := false:
if(not(has(evalf(tau),evalf(xi)))) then
  tau := ArrayTools[Append](tau,xi):
  appended_zero := true:
end if:

# make confluency vector
s:=Array([seq(1,j=1..ArrayTools[Size](tau)[2])]):
s[1] := s_t:

if(appended_zero) then
  s[ArrayTools[Size](tau)[2]] := s_t*(r+1):
  s[ArrayTools[Size](tau)[2]-1] := s_t:
else
  s[ArrayTools[Size](tau)[2]] := s_t:
  s[(ArrayTools[Size](tau)[2]+1)/2] := s_t*(r+1):
end if;

# Hermite interpolation
xy:=evalf([seq([tau[m], seq(eval(diff(f(x), [x$j]), x = tau[
   m]), j = 0 .. s[m]-1)], m = 1 .. ArrayTools[Size](tau)
   [2])]);
p:=x->add(alpha[i]*x^(i-1), i = 1 .. add(s));
interpolant_coeffs := solve({seq(seq(eval(diff(p(x), [x$i]),
    x = xy[j][1]) = xy[j][i+2],i = 0 .. s[j]-1 ), j = 1 ..
    ArrayTools[Size](tau)[2])}):
  p:=x->add(rhs(interpolant_coeffs[i])*x^(i-1), i=1..add(s));

  # Calculating the moments
  wts:=map(c->value(c),Array([seq(Int(x^i*exp(I*omega*g(x)), x
     = 0 .. 1), i = 0 .. add(s)-1)])):

  Int_f := p(0)*wts[1]+add(rhs(interpolant_coeffs[i])*wts[i],
    i = 2 .. add(s)):

  # Calculation of the integral
  return [seq(evalf(subs(omega = w, Int_f)),w=omega_range)]:

end proc:
```

A.3 Levin-Type Methods

A.3.1 Levin-Hermite Method

Python Code

```python
def Genbarywts(tau,s):
  '''
  Generates Barycentric Weights given a set of nodes tau with
  confluency vector s
  '''
  import numpy as np
  from scipy.special import factorial

  n = len(tau)
  s_max = max(s)
  d = sum(s)

  delta_tau = np.zeros([n,n])
  beta = np.zeros(n)
  [u,v,w] = [np.zeros([n,s_max]),np.zeros([n,s_max+1]),np.
     zeros([n,s_max])]

  for i in range(0,n):
    v[i,0] = 1
    for j in range(0,n):
      delta_tau[i,j] = tau[i]-tau[j]
  delta_tau = delta_tau+np.eye(n)
  delta_tau_recip = 1/(delta_tau)

  for i in range(0,n):
    delta_tau_recip[:,i] = delta_tau_recip[:,i]**s[i]

  def range_inc(a,b):
    '''
    Modified range function that will include the right end
       point in the
    return array.

    It will also return an array of size 1 if a=b (as opposed
    to an empty array returned by regular arange)
    a = lower limit
    b = upper limit
    '''
    if a==b:
      return np.full(1,a)
    else:
      return np.arange(a,b+1)

def strainer_array(i,n):
    '''
    Returns an array of size n with a 0 in the i^th position
    and 1's for all the other entries
    '''
    temp = np.full(n,1)
    temp[i] = 0
    return temp

for i in range_inc(0,n-1):
    for m in range_inc(0,s_max-1):
      u[i,m] = np.sum(strainer_array(i,n)*s*delta_tau[:,i]**(-
        m-1))

    for m in range_inc(0,s_max-1):
      v[i,m+1] = np.sum(u[i,0:m+1]*v[i,0:m+1][::-1])/(m+1)

    beta[i] = np.prod(delta_tau_recip[i,:])

    for m in range_inc(1,s[i]):
      w[i,m-1] = beta[i]*v[i,s[i]-m]


'''
Hermitian differentiation matrix calculation below
'''
D = np.zeros([d,d])
brks = np.cumsum([0]+s)
irow = 0
sum_range = np.arange(0,n)

for k in range(0,n):
    # Trivial Rows
    for j in np.arange(0,s[k]-1):
      D[irow,brks[k].astype(int)+j+1] = j+1
      irow += 1

    # Non-Trivial Rows
    for i in sum_range[sum_range!=k]:
      for j in np.arange(0,s[i]):
        g = 0
```

```
          for mu in np.arange(j,s[i]):
            g = g + w[i,mu]*(tau[k]-tau[i])**(j-1-mu)
          D[irow,brks[i].astype(int)+j] = g/w[k,s[k]-1]

      D[irow,brks[k]+1:brks[k]+s[k]] = -w[k,0:s[k]-1]/w[k,s[k
         ]-1]
      D[irow,brks[k]] = -np.sum(D[irow,brks[0:len(brks)-1]])
      D[irow,:] = D[irow,:]*s[k]
      irow += 1

    return [D,w]
def Levin_Int(F,G,tau,s,omega_range):
    # Package imports
    import sympy as sp
    import pprint
    import numpy as np
    from scipy.integrate import quad
    from scipy.special import factorial,comb
    import matplotlib.pyplot as pl
    from sympy import diff,Symbol,limit,evalf,sqrt,sin,cos,root,
       log,exp,simplify
    from genbarywts import Genbarywts
    import jeet_mod_methods as jt

    x = sp.Symbol('x')

    def f(x):
        return F(x)

    def g(x):
      return G(x)

    g_num = sp.lambdify(x,g(x))
    Levin = np.full(len(omega_range),np.complex(0,0))

    Number_Of_Nodes = len(tau)
    d = sum(s)

    # Calculating the function and derivative values of f(x) at
       the nodes
    #
    # p = [[f(t_0),f'(t_0),...,f^(s_0-1)(t_0)],...,[f(t_n),f'(
       t_n),...,f^(s_n-1)(t_n)]]
    p = np.full([Number_Of_Nodes,max(s)],np.complex(0,0))
    p_rhs = np.full([Number_Of_Nodes,max(s)],np.complex(0,0))
```

```
for i in range(0,len(tau)):
 for j in range(0,s[i]):
  p[i,j] = simplify(sp.diff(f(x),x,j).subs(x,tau[i]))/
     factorial(j)
  p_rhs[i,j] = simplify(sp.diff(f(x),x,j).subs(x,tau[i]))

p = jt.Decompress(p,s)
p_rhs = jt.Decompress(p_rhs,s)

[D,w] = Genbarywts(tau.tolist(),s.tolist())

tau_decompressed = np.zeros(d)
k = 0
for i in range(0,len(tau)):
 for j in range(0,s[i]):
  tau_decompressed[k] = tau[i]
  k+=1
u = g_num(tau_decompressed)


Sum_s = np.cumsum([0]+s.tolist())[0:-1]

i = 0
for omega in omega_range:

 # Create a Coefficient matrix for the linear system
 # f = (D + iwg')*P
 # The non-trivial work here is calculating the product
    term g'P
 Coeff_matrix = np.full([d,d],np.complex(0,0))
 k = 0
 for j in Sum_s:
  for s_i in range(0,s[k]):
   Coeff_matrix[j+s_i,:] = D[j+s_i,:]*factorial(s_i)
   for l in jt.range_inc(1,s_i):
    Coeff_matrix[j+s_i,:] = Coeff_matrix[j+s_i,:] + np.
      complex(0,omega)*comb(s_i,l)*factorial(l-1)*D[j+l
      -1,:]*diff(g(x),x,s_i-l+1).subs(x,tau[k])

   Coeff_matrix[j+s_i,:] =  Coeff_matrix[j+s_i,:]+ np.
     complex(0,omega)* np.eye(d)[j,:]*diff(g(x),x,s_i+1)
     .subs(x,tau[k])
  k+=1

 # Solving f = (D + iwg')*P
```

```
    P = np.linalg.solve(Coeff_matrix,p_rhs)

    Levin[i] = (P[-s[-1]])*np.exp(complex(0,omega)*u[-1])-P
       [0]*np.exp(complex(0,omega)*u[0])
    i+=1


return Levin
```

Maple Code

```
#
# BHIP: Barycentric Hermite Interpolation Program
#
# (c) Robert M. Corless, December 2007, August 2012
#
# Compute the barycentric form of the unique Hermite
   interpolant
# of the polynomial given by values and derivative values of
# p(t) at the nodes tau.
#
# CALLING SEQUENCES
#
#   ( p, gam ) := BHIP( flist, tau, t );
#   ( p, gam ) := BHIP( ftayl, tau, t, 'Taylor' = true, '
   Denominator' = q );
#   ( p, gam, DD ) := BHIP( ftayl, tau, t, <opts>, 'Dmat'=true
    )
#
# Processing: local Laurent series.
#             This approach is different to that of
# Reference: C. Schneider & W.Werner, "Hermite
#            Interpolation: The Barycentric Approach",
#            Computing 46, 1991, pp 35-51.
#
BHIP :=
proc( pin::list, tau::list, t::name,
      {Taylor::truefalse:=true},
      {Conditioning::truefalse:=false},
      {Dmat::truefalse:=false},
      {Denominator::{algebraic,list}:=1} )
     local brks, d, DD, denr, dens, dgam,
           dr, g, gam, ghat,h, i, irow, j,
           k, mu, n, numr, nums,
           p, P, q, r, rs, rt, s, smax, sq;

     n := nops(tau);
     if nops(pin) <> n then
        error "Mismatched size of node list and data list"
     end if;

     if nops(convert(tau,set)) < n then
        error "Nodes must be distinct, with confluency
           explicitly specified."
```

```
end if;

p := map(t -> `if`(t::list,t,[t]),pin); # singletons ok
s := map(nops,p); # confluency
smax := max(op(s));
if smax = 0 then
  error "At least one piece of data is necessary."
end if;
d := -1 + add( s[i], i=1..nops(s) );  # degree bound
p := `if`( Taylor, p, [seq([seq(p[i][j]/(j-1)!,j=1..s[i])
   ],i=1..n)] );

gam := Array( 1..n, 0..smax-1 ); #default 0
if Conditioning then
   dgam := Array( 1..n, 0..smax-1, 1..n ); #default 0
end if;

# The following works for n>=1
for i to n do
    if s[i] > 0 then # ignore empty lists
       h[i] := mul( (t-tau[j])^s[j], j = 1..i-1 )*
               mul( (t-tau[j])^s[j], j=i+1..n );
       r[i] := series( 1/h[i], t=tau[i], s[i] );
       for j to s[i] do
          gam[i,s[i]-j] := coeff( r[i], t-tau[i], j-1) ;
             #op( 2*j-1, r[i] );
       end do;
       if Conditioning then
          # We could compose a series for 1/(t-tau[k])
             with
          # what we know, but using the kernel function "
             series"
          # is likely faster.
          for k to i-1 do
             dr[i,k] := series( s[k]/h[i]/(t-tau[k]), t=
                tau[i], s[i] );
             for j to s[i] do
               dgam[i, s[i]-j, k] := coeff( dr[i,k], t-
                  tau[i], j-1 );
             end do;
          end do;
          # We could reuse earlier series, and do one O(n
             ^2)
          # computation to get gam[i,-1], but it's
             simpler to
          # use series (and likely faster because series
             is in the kernel)
          dr[i,i] := series( 1/h[i], t=tau[i], s[i]+1 );
          # We implicitly divide this by t-tau[i], and
             take
          # coefficients one higher.
          for j to s[i] do
             dgam[i,s[i]-j,i] := j*coeff( dr[i,i], t-tau[
                i], j );
          end do;
          for k from i+1 to n do
             dr[i,k] := series( s[k]/h[i]/(t-tau[k]), t=
                tau[i], s[i] );
             for j to s[i] do
               dgam[i, s[i]-j, k] := coeff( dr[i,k], t-
                  tau[i], j-1 );
             end do;
          end do;
       end if;
    end if;
end do;

if not (Denominator::algebraic and Denominator=1) then
   # adjust gam by folding in q
   if Denominator::list then
      if nops(Denominator)<>n then
         error "Denominator list (q) has the wrong length
            ."
      end if;
      q :=`if`( Taylor, q, [seq([seq(q[i][j]/(j-1)!,j=1..
         s[i])],i=1..n)] );
   else
      ghat := Array( 1..n );
      for i to n do
         sq := series(Denominator,t=tau[i],s[i]);
         ghat[i] :=[seq(coeff(sq,t-tau[i],j),j=0..s[i]-1)
            ];
      end do;
      q := [seq(ghat[i],i=1..n)];
   end if;
   ghat := Array( 1..n, 0..smax-1 );
   for i to n do
      for j from 0 to s[i]-1 do
         ghat[i,j] := add( gam[i,j+k]*q[i][k+1], k=0..s[i
            ]-j-1 );
          end do;
        end do;
        gam := ghat;
      end if;

      P := mul( (t-tau[i])^s[i],i=1..n)*
          add(add(gam[i,j]/(t-tau[i])^(1+j)*
                  add(p[i][1+k]*(t-tau[i])^k, k=0..j),
                  j=0..s[i]-1),
              i=1..n );

      # Translated from Matlab.  Nearly working.
      if Dmat then
        # Compute differentiation matrix
        DD := Matrix( d+1, d+1 );
        brks := [seq(add(s[j],j=1..i-1),i=1..nops(s))]; #cumsum
           ([0,s.']);
        irow := 0;
        for k to n do
          # trivial rows
          for j to s[k]-1 do
            irow := irow+1;
            # next available row
            DD[irow,brks[k]+j+1] := j;  # result is in Taylor
               form
          end;
          # Nontrivial row
          irow := irow+1;
          for i in [seq(j,j=1..k-1),seq(j,j=k+1..n)] do
            for j to s[i] do
              g := 0;
              for mu from j-1 to s[i]-1 do
                  g := g + gam[i,mu]*(tau[k]-tau[i])^(j-2-mu);
              end;
              DD[irow,brks[i]+j] := g/gam[k,s[k]-1];
            end;
          end;
          DD[irow,brks[k]+2..brks[k]+s[k]] := -gam[k,0..s[k
             ]-2]/gam[k,s[k]-1];
          # Final entry
          DD[irow,brks[k]+1] := -add( DD[irow,brks[j]+1], j=1..
             nops(brks) );
          DD[irow,1..-1] := DD[irow,1..-1]*s[k];  # want Taylor
              form of derivative
        end;
```

```
    end if;

    return P, gam, `if`(Conditioning,dgam,NULL), `if`(Dmat,DD
        ,NULL) ;
end proc:
read "BHIP.mpl":

unroll := proc(p,s)
  local ret_arr,i,j,k:
  ret_arr := [ ]:
  for i from 1 to nops(p) do
    for j from 1 to s[i] do
      ret_arr := [op(ret_arr),p[i,j]]:
    od:
  od:
  return ret_arr:
end proc:

Levin_Hermite := proc(F,G,tau_in,N_in,s_in,omega_range)
  local D, Levin, P, Sum_s, a, b, d, gam_t, i, j, k, l, p,
     p_rhs, p_t, s, s_i, tau, w, Coeff_matrix, num_nodes,
     tau_unrolled:
  Levin := Matrix(nops(omega_range),1,fill=0):


  num_nodes := N_in:
  a := tau_in[1]:
  b := tau_in[2]:
  tau := [seq((a+b)/2+(b-a)*cos(Pi*(num_nodes-k)/(num_nodes-1)
     )/2, k = 1 .. num_nodes)]:
  s := [seq(1,k=1..num_nodes)]:
  s[1] := s_in:
  s[num_nodes] := s_in:
  d := add(s[k], k=1 .. nops(s)):


  p := []:
  p_rhs := []:

  for i from 1 to nops(tau) do
      p := [op(p),[seq(eval(diff(F(x),[x$(j-1)]),x=tau[i])/
         factorial(j),j=1..s[i])]]:
      p_rhs := [op(p_rhs),[seq(eval(diff(F(x),[x$(j-1)]),x=tau
         [i]),j=1..s[i])]]:
  od:
```

```


( p_t, gam_t, D ) := BHIP( p, tau, t, 'Dmat'=true ):
p := unroll(p,s):
p_rhs := unroll(p_rhs,s):
tau_unrolled := Matrix(d,1,fill=0):
k := 1:
for i from 1 to num_nodes do
  for j from 1 to s[i] do
    tau_unrolled[k] := tau[i]:
    k := k + 1:
  od:
od:
Sum_s := Statistics[CumulativeSum]([0,op(s)])[1..nops(s)]:

i := 1:
for w in omega_range do
  Coeff_matrix := Matrix(d,d,fill=0):
  k := 1:
  for j in Sum_s do
    j := convert(j,rational):
    for s_i from 0 to s[k]-1 do
      Coeff_matrix[j+1+s_i] := D[j+1+s_i]*factorial(s_i):
      for l from 1 to s_i do
        Coeff_matrix[j+1+s_i] := Coeff_matrix[j+1+s_i]+I*w*D
           [j+l]*eval(diff(G(x),[x$(s_i-l+1)]),x=tau[k])*
           factorial(l-1)*combinat[numbcomb](s_i,l):
      od:
      Coeff_matrix[j+s_i+1] := Coeff_matrix[j+s_i+1] +
         Matrix(d,d,shape=identity)[j+1]*eval(diff(G(x),[x$(
         s_i+1)]),x= tau[k])*I*w:
    od:
    k:= k + 1:
  od:



P := LinearAlgebra[LinearSolve](evalf(Coeff_matrix),
   LinearAlgebra[Transpose](convert(p_rhs,Matrix))):
Levin[i] := (P[LinearAlgebra[Dimensions](P)[1]-s[nops(s)
   ]+1]*exp(I*w*G(b))-P[1]*exp(I*w*G(a))):
i := i + 1:
od:

return convert(Levin,list):
```

```
end proc:

(*

N := 4:
tau := [seq(cos(Pi*(N-k)/(N-1)), k = 1 .. N)]:
s := [seq(2,k=1..N)]:
f := x -> cos(x):
g := x -> (x):
w_range := [seq(k,k=2 .. 2)]:
# F,G,tau_in,N_in,s_in,omega_range
intg := Levin_Hermite(f,g,[-1,1],5,2,w_range):
exact_vals := [seq(evalf(int(f(x)*exp(I*w*g(x)),x=tau[1]..tau[
   N])),w=w_range)]:
err := [seq(abs(intg[k]-exact_vals[k]),k=1..nops(exact_vals))
   ]:
print(exact_vals):
print(evalf(intg)):
print(evalf(err)):
*)
```

A.3.2 Levin-Bernstein Method

```
import numpy as np
import matplotlib.pyplot as pl
from scipy.special import comb
from scipy.integrate import quad
from sympy import diff,limit,evalf,sqrt,sin,cos,root,log,exp,
   simplify,lambdify,Symbol
import jeet_mod_methods as jt
# from Hermite_Levin import Internal_Quad

def B(x,n,k,a=0,b=1):
  '''
    Input:
    x = point at which Bernstein polynomial needs to be
       evaulated
    n = degree of the polynomial
    k = k^th bernstein polynomial of degree n
    Output:
    scalar or array of real/complex numbers
  '''
  from scipy.special import comb
  return comb(n,k,exact=True)*(x-a)**k*(b-x)**(n-k)/(b-a)**n

def Coeff_to_Poly(coeff_list,x,a=0,b=1):
  '''
  "Synthesis" Operator: Given a sequence of coefficients, this
      returns the value of the
  polynomial at a point x (or for an array of points xran)

    coeff_list = array of coefficients (real or complex
       numbers)
    x      = point at which polynomial needs to be evaluated
  '''
  degree = len(coeff_list)-1
  result = 0
  for j in range(0,degree+1):
    result += coeff_list[j]*B(x,degree,j,a,b)
  return result

def Bern_Interpolate(tau,ftau,a=0,b=1):
  '''
  Interpolates the set of points {(tau,f(tau))} in the
     Bernstein Basis
  (Using the obvious approach and not a fast algorithm for
    simplicity)
  '''
  n = len(tau)-1
  [a,b] = [tau[0],tau[-1]]
  c = np.full(n+1,0)
  A = np.zeros([n+1,n+1])
  for i in range(0,n+1):
    for j in range(0,n+1):
      A[i,j] = B(tau[i],n,j,a,b)
  return np.linalg.solve(A,ftau)


def Bernstein_diff_matrix(n,a=0,b=1):
  '''
   n = order of the family of Bernstein polynomials that
     we want the diff matrix for
   (n^th order family has n+1 elements)
  '''
  D = np.zeros([n+1,n+1])
  D[0,0] = -n
  D[0,1] = n
  D[-1,-1] = n
  D[-1,-2] = -n
  for i in range(1,n):
    D[i,i] = 2*i-n
    D[i,i-1] = -i
    D[i,i+1] = n-i
  return D/(b-a)

def Mul_Operator_Matrix(f):
  '''
  This method creates the multiplication matrix for a function
      f
  '''
  n = len(f)-1
  w = np.zeros([2*n+1,n+1])
  for i in range(0,2*n+1):
    for j in range(max(0,i-n),min(n,i)+1):
      w[i,j] = comb(n,j)*comb(n,i-j)/comb(2*n,i)*f[i-j]
  return w

def Levin_Bern_int(F,G,tau,omega_range):
  '''
  Levin-Bernstein integration method
  '''
  n = len(tau)-1
  [a,b] = [tau[0],tau[-1]]

  coef_f = Bern_Interpolate(jt.Cheb_nodes(a,b,2*n+1),F(jt.Cheb_nodes(a,b,2*n+1)),a,b)
  coef_one = np.ones(n+1)
  coef_g = Bern_Interpolate(tau,G(tau),a,b)

  D = Bernstein_diff_matrix(n,a,b)
  coef_dg = np.dot(D,coef_g)

  Levin = np.full(len(omega_range),np.complex(0,0))

  k = 0
  for w in omega_range:
    A = np.dot(Mul_Operator_Matrix(coef_one),D)+np.complex(0,w)*Mul_Operator_Matrix(coef_dg)
    Augmented_A = np.column_stack((A,coef_f))
    V = np.linalg.svd(Augmented_A)[2].conj()

    if(V[-1,-1]!=0):
      sol = -1/V[-1,-1]*V[-1,:-1]

    Levin[k] = sol[-1]*np.exp(complex(0,w)*G(tau[-1]))-sol[0]*np.exp(complex(0,w)*G(tau[0]))
    k+=1
  return Levin
```

A.3.3 Levin-Compact Finite Difference (CFD)

```
def Compact_diff_matrix(tau):
# Non regularized version
  import numpy as np
  import matplotlib.pyplot as pl

  N = len(tau)
  [A,B] = [np.zeros([N,N]),np.zeros([N,N])]
  h = np.diff(tau)

  for i in range(1,N-1):
    A[i,i-1] = 1.0/(h[i]+h[i-1])**2
    A[i,i] = 1.0/h[i]**2
    A[i,i+1] = h[i-1]**2/(h[i]+h[i-1])**2/h[i]**2
    B[i,i-1] = (4*h[i-1]+2*h[i])/h[i-1]/(h[i-1]+h[i])**3
    B[i,i] = 2*(h[i-1]-h[i])/h[i-1]/h[i]**3
    B[i,i+1] = -(4*h[i]+2*h[i-1])*h[i-1]**2/(h[i]+h[i-1])**3/h[i]**3

  A[0,0] = 1/(h[0]+h[1])/(h[0]+h[1]+h[2])
  A[0,1] = 1/h[1]/(h[1]+h[2])
  B[0,0] = ((4*h[0]**2+6*h[0]*h[1]+3*h[0]*h[2]+2*h[1]**2+2*h[1]*h[2])
           /(h[0]+h[1])**2/(h[0]+h[1]+h[2])**2/h[0])
  B[0,1] = (1/h[0]*((-2*h[1]+h[0])*h[2]+2*h[1]*(-h[1]+h[0]))/
      h[1]**2/(h[1]+h[2])**2
      )
  B[0,2] = -h[0]**2/(h[1]+h[0])**2/h[1]**2/h[2]
  B[0,3] = h[0]**2/(h[2]+h[1]+h[0])**2/(h[2]+h[1])**2/h[2]

  A[-1,-2] = 1/h[-2]/(h[-2]+h[-3])
  A[-1,-1] = 1/(h[-1]+h[-2])/(h[-1]+h[-2]+h[-3])
  B[-1,-4] = -h[-1]**2/(h[-3]+h[-2]+h[-1])**2/(h[-3]+h[-2])**2/h[-3]
  B[-1,-3] = h[-1]**2/(h[-2]+h[-1])**2/h[-2]**2/h[-3]
  B[-1,-2] = (1/h[-1]*((2*h[-2]-h[-1])*h[-3]+2*h[-2]*(h[-2]-h[-1]))/h[-2]**2/
        (h[-3]+h[-2])**2
        )
  B[-1,-1] = -((4*h[-1]**2+(6*h[-2]+3*h[-3])*h[-1]+2*h[-2]*(h[-3]+h[-2]))
        /h[-1]/(h[-2]+h[-1])**2/(h[-3]+h[-2]+h[-1])**2
        )

  return [-A,B]


def Levin_Int_Comp_diff(F,G,tau,omega_range):
# Package imports
  import numpy as np
  from scipy.integrate import quad
  from scipy.special import factorial,comb
  import matplotlib.pyplot as pl
  from sympy import diff,Symbol,limit,evalf,sqrt,sin,cos,root,
     log,exp,simplify,N
  from genbarywts import Genbarywts
  import jeet_mod_methods as jt

  x = Symbol('x')

  def f(x):
      return F(x)

  def g(x):
    return G(x)

  Levin = np.full(len(omega_range),np.complex(0,0))
  Number_Of_Nodes = len(tau)

  p = np.full([Number_Of_Nodes],np.complex(0,0))
  g_matrix = np.diag(np.ones(Number_Of_Nodes))

  for i in range(0,Number_Of_Nodes):
    g_matrix[i,i] = diff(g(x),x).subs(x,tau[i])
    p[i] = f(tau[i])

  [A,B] = Compact_diff_matrix(tau)

  i = 0
  for omega in omega_range:
    # P = np.linalg.solve(B+np.complex(0,omega)*np.dot(A,
       g_matrix),np.dot(A,p))
    P = jt.TSVD_Solve(B+np.complex(0,omega)*np.dot(A,g_matrix)
       ,np.dot(A,p),1e-13)
    Levin[i] = (P[-1]*np.exp(np.complex(0,omega)*float(N(g(tau
       [-1]))))-
          P[0]*np.exp(np.complex(0,omega)*float(N(g(tau[0]))))
             )
    print(Levin[i])
    i+=1
  return Levin
```

A.4 Moment-Free Methods

A.4.1 Moment-Free Basis Verification Code

```
restart:
Digits := 15:

# The expression for phi. The basis elements are given by L(
   phi). We want to check whether the "nicer" expression,
   L_phi, given for L(phi) holds.

L := (F,G) -> diff(F,x)+I*w*diff(G,x)*F;

_Envsignum0 := 0:
phi := (x_t,r_t,k_t,g_t)->eval(piecewise(x_t<0,piecewise(modp(
   r_t,2)=0,(-1)^k_t,modp(r_t,2)=1,(-1)^k_t*exp(-(1+k_t)/r_t*I
   *Pi)),x_t>=0,-1)*w^(-(k_t+1)/r_t)/r*exp(-I*w*g_t+(1+k_t)
   /(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-I*w*g_t)-GAMMA((1+k_t)/
   r_t,0)),x=x_t);

# The "nicer" expression in question
L_phi := (x_t,r_t,k_t,g_t)-> signum(x_t)^(r_t+k_t+1)*abs(g_t)
   ^((k_t+1)/r_t-1)*diff(g_t,x)/r_t;

# The pieces of piecewise phi
phi_l_odd := (x_t,r_t,k_t,g_t)->eval((-1)^k_t*exp(-(1+k_t)/r_t
   *I*Pi)*w^(-(k_t+1)/r_t)/r*exp(-I*w*g_t+(1+k_t)/(2*r_t)*I*Pi
   )*(GAMMA((1+k_t)/r_t,-I*w*g_t)-GAMMA((1+k_t)/r_t,0)),x=x_t)
   ;
phi_l_even := (x_t,r_t,k_t,g_t)->eval((-1)^k_t*w^(-(k_t+1)/r_t
   )/r*exp(-I*w*g_t+(1+k_t)/(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-
   I*w*g_t)-GAMMA((1+k_t)/r_t,0)),x=x_t);
phi_r := (x_t,r_t,k_t,g_t)->eval(-w^(-(k_t+1)/r_t)/r*exp(-I*w*
   g_t+(1+k_t)/(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-I*w*g_t)-
   GAMMA((1+k_t)/r_t,0)),x=x_t);

# Simplify the basis elements expression with a fixed g(x).
   This g(x) has a stationary point of order r+1 at x=0.
# A few notes to anyone playing with this code:
# try
# g := x -> -x^r:
# The compact expression doesn't match L(phi) in this case. So
    it is likely that the assumption eval(diff(g(x),[x$r]),x
   =0)>0 is used to gain this compact expression.

```

```
r := 2:
g := x -> x^r:

assume(w::real):
additionally(w>1):

odd_exp := simplify(L(phi_l_odd(x,r,k,g(x)),g(x)));
even_exp := simplify(L(phi_l_even(x,r,k,g(x)),g(x)));
right_exp := simplify(L(phi_r(x,r,k,g(x)),g(x)));

if(not modp(r,2)=0) then

  for k from 1 to 10 do
    p[k] := plot(L_phi(x,r,k,g(x)),x=-1..0,color=green,title =
        "Compact expression" ):
    q[k] := plot(odd_exp,x=-1..0,color=red, title = "L(phi)
       expression" ):
    b[k] := plot(odd_exp-L_phi(x,r,k,g(x)),x=-1..0,color=red,
       style = point):
  od:
  plots[display](seq({p[k]},k=1..10));
  plots[display](seq({q[k]},k=1..10));
  plots[display](seq({b[k]},k=1..10));

else

  for k from 1 to 10 do
    p[k] := plot(L_phi(x,r,k,g(x)),x=-1..0,color=green, title
       = "Compact Expression" ):
    q[k] := plot(even_exp,x=-1..0,color=red, title = "L(phi)
       expression" ):
    b[k] := plot(even_exp-L_phi(x,r,k,g(x)),x=-1..0,color=red,
        style = point , title = "Residual"):
  od:
  plots[display](seq({p[k]},k=1..10));
  plots[display](seq({q[k]},k=1..10));
  plots[display](seq({b[k]},k=1..10));

end if;

for k from 1 to 10 do
  p[k] := plot(L_phi(x,r,k,g(x)),x=0..1,color=green, title = "
     Compact Expression" ):
  q[k] := plot(right_exp,x=0..1,color=red, title = "L(phi)
     expression" ):
```

```
  b[k] := plot(right_exp-L_phi(x,r,k,g(x)),x=0..1,color=red,
     style = point , title = "Residual"):
od:
plots[display](seq({p[k]},k=1..10));
plots[display](seq({q[k]},k=1..10));
plots[display](seq({b[k]},k=1..10));
```

A.4.2 Moment-Free Asymptotic Method

```
Calc_Coefficients := proc(f_t,g_t,r_t::numeric)
  # Computes the interpolation coefficients by solving a
     system of equations

  local eq_sys,sols,c,rhs_eqs,L_phi:

  phi := (x_t,r_t,k_t,g_t)->eval(piecewise(x<0,piecewise(modp(
     r_t,2)=0,(-1)^k_t,modp(r_t,2)=1,(-1)^k_t*exp(-(1+k_t)/r_t
     *I*Pi)),x>0,-1)*w^(-(k_t+1)/r_t)/r*exp(-I*w*g_t+(1+k_t)
     /(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-I*w*g_t)-GAMMA((1+k_t)
     /r_t,0)),x=x_t);

  L_phi := (x_t,r_t,k_t,g_t)-> signum(x_t)^(r_t+k_t+1)*abs(g_t
     )^((k_t+1)/r_t-1)*diff(g_t,x)/r_t;

  rhs_eqs := [seq((diff(f_t,[x$j])),j=0..r_t-2)];

  rhs_eqs := map(h->limit(h,x=0),rhs_eqs);
  eq_sys := [seq(diff( add( c[k+1]*L_phi(x,r_t,k,g_t),k=0..r_t
     -2 ) ,[x$j]),j=0..r_t-2)];
  eq_sys := map(h->limit(h,x=0),eq_sys);
  sols := solve({seq(eq_sys[j+1]=rhs_eqs[j+1],j=0..r_t-2)},{
     seq(c[j+1],j=0..r_t-2)});

  return [seq(rhs(sols[i]),i=1..nops(sols))]:

end proc:


Moment_free_asymp := proc(F,G,tau_in::list,xi,r,p,w_range)

  local L_phi,mu_f,L,a,b,f,g,xi_t,sig_coef,sigma,func_arr_l,
     func_arr_r,L_func_arr_l,L_func_arr_r,sigma_arr_l,
     sigma_arr_r,l_term_2,r_term_2,sum_terms_2,l_term_1,
     r_term_1,sum_terms_1,phi:

  Digits := 16:
```

```
_Envsignum0 := 0:

# Transforming the integral to the correct interval
  a := tau_in[1]:
  b := tau_in[2]:
f := x -> F(x*(b-a)/2+(b+a)/2):
  g := x -> G(x*(b-a)/2+(b+a)/2):
  xi_t := (2*xi-a-b)/(b-a):

  # basis functions
phi := (x_t,r_t,k_t,g_t)->eval(piecewise(x<0,piecewise(modp(
   r_t,2)=0,(-1)^k_t,modp(r_t,2)=1,(-1)^k_t*exp(-(1+k_t)/r_t
   *I*Pi)),x>0,-1)*w^(-(k_t+1)/r_t)/r*exp(-I*w*g_t+(1+k_t)
   /(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-I*w*g_t)-GAMMA((1+k_t)
   /r_t,0)),x=x_t);

L_phi := (x_t,r_t,k_t,g_t)-> signum(x_t)^(r_t+k_t+1)*abs(g_t
   )^((k_t+1)/r_t-1)*diff(g_t,x)/r_t;

mu_f := (c,r_t,x,g_t) -> add(c[k+1]*phi(x,r_t,k,g_t),k=0..
   r_t-2);

L := (F,G) -> diff(F,x)+I*w*diff(G,x)*F;

# Computing the series
sigma[0] := f(x):

for k from 1 to p do
    sig_coef[k-1] := Calc_Coefficients(sigma[k-1],g(x),r):
    sigma[k] := diff( (sigma[k-1] - add( sig_coef[k-1][j+1]*
       L_phi(x,r,j,g(x)),j=0..r-2 ))/diff(g(x),x) ,x);
od:
sig_coef[p] := Calc_Coefficients(sigma[p],g(x),r):

func_arr_r := [ seq(phi(1,r,k,g(x))*exp(I*w*g(1)),k=0..r-2)
   ]:
func_arr_l := [ seq(phi(-1,r,k,g(x))*exp(I*w*g(-1)),k=0..r
   -2) ]:

L_func_arr_l := [ seq(eval(L_phi(-1,r,k,g(x)),x=-1),k=0..r
   -2) ]:
L_func_arr_r := [ seq(eval(L_phi(1,r,k,g(x)),x=1),k=0..r-2)
   ]:

sigma_arr_l := [ seq( eval(sigma[k],x=-1) ,k=0..p) ]:
  sigma_arr_r := [ seq( eval(sigma[k],x=1) ,k=0..p) ]:

  l_term_2 := [ seq((sigma_arr_l[k+1] - add(sig_coef[k][j+1]*
     L_func_arr_l[j+1],j=0..r-2))*exp(I*w*g(-1))/eval(diff(g(x
     ),x),x=-1),k=0..p) ]:

  r_term_2 := [ seq((sigma_arr_r[k+1] - add(sig_coef[k][j+1]*
     L_func_arr_r[j+1],j=0..r-2))*exp(I*w*g(1))/eval(diff(g(x)
     ,x),x=1),k=0..p) ]:

  sum_terms_2 := r_term_2 - l_term_2:

  l_term_1 := [ seq(add(sig_coef[k][j+1]*func_arr_l[j+1],j=0..
     r-2),k=0..p) ]:
  r_term_1 := [ seq(add(sig_coef[k][j+1]*func_arr_r[j+1],j=0..
     r-2),k=0..p) ]:

  sum_terms_1 := r_term_1 - l_term_1:

  Q_A := add(1/(-I*w)^k*sum_terms_1[k+1],k=0..p)-add(1/(-I*w)
     ^(k+1)*sum_terms_2[k+1],k=0..p):

  return [ (b-a)/(2)*seq(evalf(eval(Q_A,w=w_range[j])),j=1..
     nops(w_range)) ]:

  eval(vals)

end proc:
```

A.4.3 Moment-Free Filon Method

```
Moment_free_filon := proc(F,G,tau_in::list,xi,r,s_in,N_in,
   w_range)
  local L_phi, phi, N, Q_F, a, b, f, f_ap, g, i, n, varphi, s,
      s_t, tau, vals, xi_t, xy, func_arr_l, func_arr_r,
     int_coeffs, s_end_pts, s_stat_pt, stat_pt_pos,
     interpolant_coeffs_free_var:

    Digits := 16:
    _Envsignum0 := 0:

    # Transforming the integral to [-1,1]
    a := tau_in[1]:
    b := tau_in[2]:
  f := x -> F(x*(b-a)/2+(b+a)/2):
    g := x -> G(x*(b-a)/2+(b+a)/2):
```

```
    xi_t := (2*xi-a-b)/(b-a):

    # setting the confluencies appropriately
    N := N_in:
    s_t := 1:
    s_end_pts := s_in:
    s_stat_pt := (2*s_end_pts-1)*(r-1):
    tau := Array([seq((cos(Pi*i/(N-1))), i = N-1 .. 0, -1)]);

    # Adding stat point as a node if it is not a node already
    if(not(has(evalf(tau),xi_t))) then
        tau := sort((ArrayTools[Append](tau,xi_t)):
    end if:

    stat_pt_pos := 2:
    for i from 2 to nops(tau)-1 do
        if(evalf(tau[i])=xi_t) then
            break:
        else
            stat_pt_pos := stat_pt_pos + 1:
        end if
    od:

    # make confluency vector
    N := ArrayTools[Size](tau)[2]:
    s := Array([seq(s_t,j=1..N)]):
    s[stat_pt_pos] := s_stat_pt:
    s[1] := s_end_pts:
    s[N] := s_end_pts:
    n := add(s[i],i=1..N);

    # Basis functions and interpolation
    phi := (x_t,r_t,k_t,g_t)->eval(piecewise(x<0,piecewise(
       modp(r_t,2)=0,(-1)^k_t,modp(r_t,2)=1,(-1)^k_t*exp(-(1+
       k_t)/r_t*I*Pi)),x>0,-1)*w^(-(k_t+1)/r_t)/r_t*exp(-I*w*
       g_t+(1+k_t)/(2*r_t)*I*Pi)*(GAMMA((1+k_t)/r_t,-I*w*g_t)-
       GAMMA((1+k_t)/r_t,0)),x=x_t);

    L_phi := (x_t,r_t,k_t,g_t) -> signum(x_t)^(r_t+k_t+1)*abs(
       g_t)^((k_t+1)/r_t-1)*diff(g_t,x)/r_t:

    f_ap := x -> add(c[j]*L_phi(x,r,j-1,g(x)),j=1..n):

    xy:=evalf([seq([tau[m], seq(eval(diff(f(x), [x$j]), x =
       tau[m]), j = 0 .. s[m]-1)], m = 1 .. N)]):
```

```

    interpolant_coeffs_free_var := fsolve([seq(seq(limit(diff(
       f_ap(x), [x$i]), x = xy[j][1],right) = xy[j][i+2],i = 0
        .. s[j]-1 ), j = 1 .. N)]);

    int_coeffs := [seq(rhs(interpolant_coeffs_free_var[j]),j
       =1..n)];

    # Evaluation of the integral
    func_arr_r := [ seq(phi(1,r,k,g(x))*exp(I*w*g(1)),k=0..n
       -1) ]:
    func_arr_l := [ seq(phi(-1,r,k,g(x))*exp(I*w*g(-1)),k=0..n
       -1) ]:

    Q_F := add(int_coeffs[j+1]*(func_arr_r[j+1]-func_arr_l[j
       +1]),j=0..n-1):

    vals := [(b-a)/2*seq(evalf(eval(Q_F,w=w_range[j])),j=1..
       nops(w_range))]:

    return vals:

end proc:
```

A.5 Hybrid Method

```
Digits := 16:

# Setup the problem
f := x -> 1:
g := x -> -(x-0.1)^2:
a := 0:
b := Pi:
w_range := [seq(k,k=10..100,1)]:

# Finding all the stationary points
stat_points_set := {(solve(diff(g(x),x),x))}:

stat_points := []:
int_limits := []:

# Separating the points that lie within [a,b] from the rest
for i from 1 to nops(stat_points_set) do
    if(normal(Im(stat_points_set[i]))=0) then
        if(evalf(stat_points_set[i]-b)<0 and evalf(
            stat_points_set[i]-a)>0) then
        stat_points := [op(stat_points), stat_points_set[i]]
        end if:
    end if:
od:
stat_points := sort(evalf(stat_points)):

# Finding the order of each of the stationary points
orders := [seq(2,i=1..nops(stat_points))]:
for i from 1 to nops(stat_points) do
    for j from 2 to 15 do
        if(eval(diff(g(x),[x$j]),x=stat_points[i])=0) then
            orders[i] := orders[i]+1:
        else
            break:
        end if:
    od:
od:

# Checking whether the second derivative vanishes away from
   the stationary points
second_deriv_zero_set := [op({solve(diff(g(x),[x$2])=0)})]:
real_second_deriv_zero_set := []:

if(second_deriv_zero_set = []) then
    # If the second derivative vanishes everywhere in [a,b],
       continue
    real_second_deriv_zero_set := []:
else
    for i from 1 to nops(second_deriv_zero_set) do
    # use evalf on any sols that have a RootOf
    if(type(second_deriv_zero_set[i],RootOf)) then
        second_deriv_zero_set[i] := evalf(allvalues(
            second_deriv_zero_set[i])):
    end if:
        # Ignore all the points which have a non-zero complex
           part
       if(Im(second_deriv_zero_set[i])=0) then
            # Add a point to the list if the second derivative
                vanishes but the first doesn't
            if(not(member(second_deriv_zero_set[i],stat_points
                ))) then
                if(evalf(second_deriv_zero_set[i])>a and evalf
                    (second_deriv_zero_set[i])<b) then
                     real_second_deriv_zero_set := [op(
                      real_second_deriv_zero_set),
                      second_deriv_zero_set[i]]:
                end if:
            end if:
        end if:
    od:
end if:
real_second_deriv_zero_set := sort(real_second_deriv_zero_set)
   :

# If there are no stationary points in the interval, use Levin
   -Hermite on the entire interval.
if(nops(stat_points)=0) then
    int_limits := []:
else
    # For each stationary point, make a list of
    # 1. stationary points to its left
    # 2. stationary points to its right
    # 3. and 4. The same but for points where the second
       derivative vanishes
    for i from 1 to nops(stat_points) do
        Left_stat_point_set := []:
        Right_stat_point_set := []:
        Left_second_deriv_zero_set := []:
        Right_second_deriv_zero_set := []:
        Left_interesting_points := []:
        Right_interesting_points := []:


        for j from 1 to nops(real_second_deriv_zero_set) do
            if(not normal(real_second_deriv_zero_set[j]) =
               normal(stat_points[i])) then
                if(normal(real_second_deriv_zero_set[j])<
                   normal(stat_points[i])) then
                    Left_second_deriv_zero_set := [op(
                      Left_second_deriv_zero_set),
                      real_second_deriv_zero_set[j]]:
                else
                    Right_second_deriv_zero_set := [op(
                      Right_second_deriv_zero_set),
                      real_second_deriv_zero_set[j]]:
                end if:
            end if:
        od:

```

```
        for j from 1 to nops(stat_points) do
            if(not (i=j)) then
                if(normal(stat_points[j])<stat_points[i]) then
                    Left_stat_point_set := [op(
                       Left_stat_point_set),stat_points[j]]:
                else
                    Right_stat_point_set := [op(
                       Right_stat_point_set),stat_points[j]]:
                end if:
            end if:
        od:

        Left_second_deriv_zero_set := convert(sort(
           Left_second_deriv_zero_set),float):
        Right_second_deriv_zero_set := convert(sort(
           Right_second_deriv_zero_set),float):
        Left_stat_point_set := convert(sort(
           Left_stat_point_set),float):
        Right_stat_point_set := convert(sort(
           Right_stat_point_set),float):

  # Concatenate the lists into two lists as follows:
  # 1. stationary points on the left + points where second
     derivative vanishes to the left
  # 2. Same but for the right side
        Left_interesting_points := {op(
           Left_second_deriv_zero_set),op(Left_stat_point_set)
           }:
        Right_interesting_points := {op(
           Right_second_deriv_zero_set),op(
           Right_stat_point_set)}:

  # Take a small enough symmetric interval around each
     stationary point which excludes all the points in the
     list
        dist_left := 0:
        dist_right := 0:
        if(Left_interesting_points = {}) then
            Lower_limit := a:
            dist_left := abs(stat_points[i]-a):
        else
            Lower_limit := (max(Left_interesting_points)+
               stat_points[i])/2:
            dist_left := abs(stat_points[i]-max(
               Left_interesting_points)):
```

```
        end if:

        if(Right_interesting_points = {}) then
            Upper_limit := b:
            dist_right := abs(stat_points[i]-b):
        else
            Upper_limit := (stat_points[i]+min(
               Right_interesting_points))/2:
            dist_right := abs(stat_points[i]-min(
               Right_interesting_points)):
        end if:

        radius_interval := min(dist_right,dist_left)/2:
        int_limits := [op(int_limits),[stat_points[i]-
           radius_interval,stat_points[i]+radius_interval]]:
    od:
end if:

# Make a list of intervals which have not already been
   included
levin_int_limits := []:
if(int_limits = []) then
    levin_int_limits := [[a,b]]:
else
    if( not int_limits[1][1] = a ) then
        levin_int_limits := [[a,int_limits[1][1]]]:
    end if:

    for i from 1 to nops(int_limits)-1 do
        if(not int_limits[i][2] = int_limits[i+1][1]) then
            levin_int_limits := [op(levin_int_limits),[
               int_limits[i][2],int_limits[i+1][1]]]:
        end if:
    od:

    if(not int_limits[nops(int_limits)][2] = b) then
        levin_int_limits := [op(levin_int_limits),[int_limits[
           nops(int_limits)][2],b]]:
    end if:
end if:


printf("The stationary points which lie in the integration
   domain are \n"):
print(stat_points):
```

```
printf("The orders of the stationary points are \n"):
print(orders):
printf("Integration limits over with Moment free filon needs
   to be used are:\n"):
print(int_limits):
printf("Integration limits over with Levin integration is to
   be used are:\n"):
print(levin_int_limits):

read "Levin_Hermite_Maple.mpl":
read "Moment_free_Filon.mpl":
read "filon_stat_point_proc.mpl":

int_g := [seq(0,k=1..nops(w_range))]:


for i from 1 to nops(stat_points) do
    print("Filon Method Called"):
    if(not evalf(g(stat_points[i]))=0) then
        st_pt := stat_points[i]:
        g_norm := x -> g(x)-g(st_pt):
            if(diff(g(x),[x$r])<0) then
                temp := Filon_Method_Stat_Pts(f,g_norm,
                   int_limits[i][1],int_limits[i][2],5,2,
                   stat_points[i],orders[i]-1,1,w_range):
            else
                temp := Moment_free_filon(f,g_norm,int_limits[
                   i],stat_points[i],orders[i],2,5,w_range):
            end if:
        int_g := int_g + [seq(temp[k]*exp(I*w_range[k]*g(st_pt
           )),k=1..nops(w_range))]:
    else
        if(diff(g(x),[x$r])<0) then
            int_g := int_g + Filon_Method_Stat_Pts(f,g,
               int_limits[i][1],int_limits[i][2],5,2,
               stat_points[i],orders[i]-1,1,w_range):
        else
            int_g := int_g + Moment_free_filon(f,g,int_limits[
               i],stat_points[i],orders[i],2,4,w_range);
        end if:
    end if:
od:

for i from 1 to nops(levin_int_limits) do
    int_g := int_g + Levin_Hermite(f,g,levin_int_limits[i
        ],5,2,w_range):
od:
```

Curriculum Vitae

Name:	Jeet Trivedi
Post-Secondary Education and Degrees:	University of Western Ontario London, Ontario 2017 - 2019 MSc. Applied Mathematics
	Trent University Peterborough, Ontario 2013 - 2017 Honors BSc. Mathematical Physics
Honours and Scholarships	Western Graduate Research Scholarship Trent International Global Citizenship Full Tuition Scholarship