QuadgF, an Octave subroutine to integrate strongly oscillatory functions.

September 23, 2019

1 Introduction

In order to perform diffraction calculations it is necessary to integrate numerically functions of type

$$\int f(x) \exp\{i \cdot k \cdot g(x)\}\, dx$$

with values of parameter k of the order of 10^4, which gives rise to a strongly oscillating integrant, that the standard squares of octave do not handle properly. As I didn't find any octave/matlab function already made to integrate strongly oscillatory functions, I decided to write one myself.

Browsing the bibliography on the subject, I found an article by L.F. Shampine, reference [1], which describes an algorithm that seemed easy to implement to me. In fact, the proposed algorithm is an improvement of the SSP method [2, 3], which possibly Mr. Shampine did not know, since he does not mention it in his bibliography.

In his article, Professor Shampine talks about a function he implemented, `quadgF`, based on the algorithm, but as much as I looked for it I couldn't find it -quadgF.m-. So I implemented it as well as I could, trying to vectorize it as much as possible, and kept the name used in [1].

1.1 Function use:

- `[q, num_iter] = quadgF(f, g, limI, limS)`
- `[q, num_iter] = quadgF(f, g, limI, limS, tol)`
- `[q, num_iter] = quadgF(f, g, limI, limS, tol, max_iter)`

Input variables:

- f and g are function handles. Both functions must be vectorized and return a vector of output values when given a vector of input values
- *lim_I* and *lim_S* are the lower and upper limits of integration. Both limits must be finite.
- The optional argument *tol* defines the absolute tolerance with which to perform the integration. The default value is 1e-6.
- The optional argument *max_iter* defines the maximum number of iterations the principal loop performs. The default is 5. The user can set a higher number, but a very large number of *num_iter* means the algoritm used in this function is not suitable for calculating the integral.

Output variables

- The result of the integration is returned in q, a scalar comples or real number.
- The optional output *num_iter*, a positive integer, indicates the number of iterations the principal loop performs.

2 Algorithm

The quadgF function evaluates numerically integrals of type:

$$I = \int_{limI}^{limS} f(x) \exp\{ig(x)\}\, dx \tag{1}$$

by means of an adaptive, strongly vectorized quadrature described in [1]. To do this, it divides the integration interval, $[limI, limS]$ into sub-intervals and approximates the functions $f(x)$ and $g(x)$ by cubic splines in each of the sub-intervals:

$$I \approx \sum_{m=1}^{N} Q_m = \sum_{m=1}^{N} \int_{x_m}^{x_{m+1}} S(x)\, \mathrm{e}^{is(x)} dx = \sum_{m=1}^{N} \int_{0}^{h} S(t)\, \mathrm{e}^{is(t)} dt \tag{2}$$

being: $h = x_{m+1} - x_m$, $t = x - x_m$ and

$$S(t) = b_1 + b_2 t + b_3 t^2 \qquad s(x) = a_1 + a_2 t + a_3 t^2 \tag{3}$$

where the coefficients a_i and b_i are given in each sub-intervals by:

$$\begin{aligned} b_1 &= f_m \qquad b_2 = \frac{4f_{m+1/2} - 3f_m - f_{m+1}}{h} \qquad b_3 = \frac{2f_m - 4f_{m+1/2} + 2f_{m+1}}{h} \\ a_1 &= g_m \qquad a_2 = \frac{4g_{m+1/2} - 3g_m - g_{m+1}}{h} \qquad a_3 = \frac{2g_m - 4g_{m+1/2} + 2g_{m+1}}{h} \end{aligned} \tag{4}$$

being $f_m = f(x_m)$, $g_{m+1/2} = g\left(x_{m+1/2}\right) \dots$

2.1 Size of sub-intervals and iteration

In principle the integration range $[limI, limS]$ is divided into 32 sub-intervals of the same width. In each sub-intervals 5 equi-spaced points are considered: $x_m, x_{m+1/4}, x_{m+1/2}, x_{m+3/4}, x_{m+1}$ in which the functions f and g are evaluated: $f_m, f_{m+1/4}, f_{m+1/2}, f_{m+3/4}, f_{m+1}$ and $g_m, g_{m+1/4}, g_{m+1/2}, g_{m+3/4}, g_{m+1}$. The last element of these quintets is the first element of the following quintet, corresponding to the following sub-interval, so that the total of points initially considered is 129 = 32-4 + 1.

With the elements 1, 3 and 5 of each quintet the coefficients of the splines are calculated, eq. 4. Whereas elements 2 and 4 serve to estimate the error that is made in approximating the f and g functions by cubic splines. Thus, using the notation: $H_f(x) = [f(x) - S(x)]^2$, the absolute error made by using the approximation in a given sub-interval can be approached by:

$$\int_{x_m}^{x_{m+1}} H_f(x)\, dx \approx \frac{256}{945} h \left[H_f\left(x_{m+1/4}\right) + H_f\left(x_{m+3/4}\right)\right] \tag{5}$$

and the condition to consider that the sub-interval is well approximated by the spline is:

$$\int_{x_m}^{x_{m+1}} H_f(x)\, dx \le \mathrm{tol}^2 \left\|f(x)\right\|^2 \frac{h}{b-a} \tag{6}$$

A reasonable estimate of the general size of $f(x)$ is obtained by applying Simpson's rule on each sub-interval:

$$\left\|f(x)\right\|^2 = \sum_{m=1}^{32} \int_{x_m}^{x_{m+1}} f^2(x)\, dx = \sum_{m=1}^{32} \frac{x_{m+1} - x_m}{3} \left[f_m^2 + 4f_{m+1/4}^2 + 2f_{m+1/2}^2 + 4f_{m+3/4}^2 + f_{m+1}^2\right].$$

For the sub-intervals where the condition 6 is met, the integral is calculated as explained in section 2.2, and the results are added to the Q figure. Meanwhile, the other intervals are saved for the next iteration.

In the next iteration the intervals that remain to be integrated are divided in two equal parts. The superscript 0 is used to denote the old subinterval, while the two new ones are denoted by 1 and 2, ex. $h^1 = h^2 = \frac{1}{2}h^0$. Now the quintets of values corresponding to each sub-interval are made up, taking advantage of the values that were already evaluated and calculating the missing ones. So for the values of variable x:

$$
\begin{array}{llll}
x_m^1 = x_m^0 & & x_m^2 = x_{m+1/2}^0 & \\
& x_{m+1/4}^1 = x_m^0 + \frac{h^1}{4} & & x_{m+1/4}^2 = x_{m+1/2}^0 + \frac{h^2}{4} \\
x_{m+1/2}^1 = x_{m+1/4}^0 & & x_{m+1/2}^2 = x_{m+3/4}^0 & \\
& x_{m+3/4}^1 = x_m^0 + \frac{3h^1}{4} & & x_{m+3/4}^2 = x_{m+3/4}^0 + \frac{h^2}{4} \\
x_{m+1}^1 = x_{m+1/2}^0 & & x_{m+1}^2 = x_{m+1}^0 &
\end{array}
$$

for the values of function $f(x)$:

$$
\begin{array}{llll}
f_m^1 = f_m^0 & & f_m^2 = f_{m+1/2}^0 & \\
& f_{m+1/4}^1 = f\left(x_{m+1/4}^1\right) & & f_{m+1/4}^2 = f\left(x_{m+1/4}^2\right) \\
f_{m+1/2}^1 = f_{m+1/4}^0 & & f_{m+1/2}^2 = f_{m+3/4}^0 & \\
& f_{m+3/4}^1 = f\left(x_{m+3/4}^1\right) & & f_{m+3/4}^2 = f\left(x_{m+3/4}^2\right) \\
f_{m+1}^1 = f_{m+1/2}^0 & & f_{m+1}^2 = f_{m+1}^0 &
\end{array}
$$

and in a similar way for the values of function $g(x)$.

The values of the spline coefficients are then calculated using eq. 4, and it is checked which of the new intervals verify condition 6. For the sub-intervals that verify this condition, the value of the integral is calculated, and the results are added to the Q figure. The sub-intervals in which the condition does not occur are moved to the next iteration. And so until no sub-intervals remain that do not meet 6, or until the maximum number of iterations `max_iter` is reached.

2.2 Integrating the sub-intervals

2.2.1 Approximation of the phase by a constant

If in a given sub-interval the coefficients a_3 and a_2 are extremely small, one can approach $s(x) \approx a_1 = cte$, in which case the quadrature of the sub-intervals is:

$$
Q_m = \mathrm{e}^{ia_1} \int_0^h \left[b_1 + b_2 t + b_3 t^2\right] dt = \mathrm{e}^{ia_1} \left[b_1 h + b_2 \frac{h^2}{2} + b_3 \frac{h^3}{3}\right] \tag{7}
$$

This quadrature is used when

$$
\|s(t) - a_1\| \leq \tau \|s(t)\| \tag{8}
$$

being τ a small enough number, an order of magnitude less than the tolerance of `quadgF`. Taking into account 3,

$$
\|s(t)\|^2 = \int_0^h \left(a_1 + a_2 t + a_3 t^2\right)^2 dt = a_1^2 h + 2a_1 a_2 \frac{h^2}{2} + (2a_1 a_3 + a_3) \frac{h^3}{3} + 2a_2 a_3 \frac{h^4}{4} + a_3^2 \frac{h^5}{5} \tag{9}
$$

and

$$
\|s(t) - a_1\|^2 = \int_0^h \left(a_2 t + a_3 t^2\right)^2 dt = a_2^2 \frac{h^3}{3} + 2a_2 a_3 \frac{h^4}{4} + a_3^2 \frac{h^5}{5} \tag{10}
$$

2.2.2 Linear phase approximation

If in a given sub-interval the coefficient a_3 is very small, one can use the approximation $s(t) \approx a_1 + a_2 t$, and in this case the quadrature of the sub-intervals is given by:

$$
Q_m = \mathrm{e}^{ia_1} \int_0^h \left[b_1 + b_2 t + b_3 t^2\right] \mathrm{e}^{ia_2 t}\, dt \tag{11}
$$

Now:

- $\int_0^h \mathrm{e}^{ia_2 t}\, dt = \dfrac{\mathrm{e}^{ia_2 t} - 1}{ia_2}$
- $\displaystyle\int_0^h t\mathrm{e}^{ia_2 t}\, dt = \left[(1 - ia_2 h)\,\mathrm{e}^{ia_2 h} - 1\right] / a_2^2$

- $\int_0^h t^2 \mathrm{e}^{ia_2 t}\,dt = \left[\left(2i + 2a_2 h - ia_2^2 h^2\right)\mathrm{e}^{ia_2 h} - 2i\right]/a_2^3$

so the quadrature can be calculated:

$$Q_m = \mathrm{e}^{ia_1}\left[b_1 I_1 + b_2 I_2 + b_3 I_3\right] \tag{12}$$

evaluating sucessively:

$$I_1 = \frac{\mathrm{e}^{ia_2 t} - 1}{i\,a_2}, \quad I_2 = \frac{h\mathrm{e}^{ia_2 t} - I_1}{i\,a_2}, \quad I_3 = \frac{h^2\mathrm{e}^{ia_2 t} - 2I_2}{i\,a_2}.$$

This approach is used when

$$\|s\left(t\right) - \left(a_1 + a_2 t\right)\| \le \tau\,\|s\left(t\right)\| \tag{13}$$

being

$$\|s\left(t\right) - \left(a_1 + a_2 t\right)\|^2 = \int_0^h a_3^2 t^4 dt = a_3^2\frac{h^5}{5}. \tag{14}$$

2.2.3 Cubic phase approximation

In the general case a_3 cannot be considered as zero. In this instance, it is introduced the variable $y = t + c_2$, $c_2 = a_2/2a_3$, so that:

$$s\left(y\right) = a_1 + a_2\left(y - \frac{a_2}{2a_3}\right) + a_3\left(y - \frac{a_2}{2a_3}\right)^2 = \left(a_1 - \frac{a_2^2}{4a_3}\right) + \left(a_2 - a_2\right)y + a_3 y^2 = c_1 + c_3 y^2 \tag{15}$$

with $c_1 = \left(a_1 - a_2^2/4a_3\right)$ and $c_3 = a_3$.

Futhermore

$$S\left(y\right) = b_1 + b_2\left(y - c_2\right) + b_3\left(y - c_2\right)^2 = d_1 + d_2 y + d_3 y^2$$

with

$$d_1 = b_1 - \frac{b_2 a_2}{2a_3} + \frac{b_3 a_2^2}{4a_3^2}, \quad d_2 = b_2 - \frac{a_2}{a_3}, \quad d_3 = b_3\,.$$

Then, the quadrature is written as:

$$Q_m = \int_{c_2}^{h+c_2}\left(d_1 + d_2 y + d_3 y^2\right)\,\exp\left\{i\left(c_1 + c_3 y^2\right)\right\}\,dy = \mathrm{e}^{ic_1}\sum_{j=1}^{3} d_j\int_{c_2}^{h+c_2} y^{j-1}\mathrm{e}^{ic_3 y^2}dy$$

While both Shampine and Stamnes use Fresnel integrals to solve the previous quadrature, I have preferred to use the complex error function, thus:

- $\int \exp\left\{ic_3 y^2\right\}\,dy = -\frac{1}{2}\sqrt{\frac{i\pi}{c_3}}\mathrm{erf}\left[-\sqrt{-ic_3}y\right]$
- $\int y\exp\left\{ic_3 y^2\right\}\,dy = \frac{-i}{2c_3}\exp\left\{ic_3 y^2\right\}$
- $\int y^2\exp\left\{ic_3 y^2\right\}\,dy = \frac{-i}{4c_3^{3/2}}\left[\sqrt{i\pi}\mathrm{erf}\left(-\sqrt{-ic_3}y\right) + 2\sqrt{c_3}y\exp\left\{ic_3 y^2\right\}\right]$

Now, using notation:

$$v\left(y\right) = \mathrm{erf}\left[-\sqrt{-ic_3}y\right] \qquad w\left(y\right) = \exp\left\{ic_3 y^2\right\}$$

we rewrite the previous integrals as:

- $\int \exp\left\{ic_3 y^2\right\}\,dy = -\frac{1}{2}\sqrt{\frac{i\pi}{c_3}}v\left(y\right)$
- $\int y\exp\left\{ic_3 y^2\right\}\,dy = \frac{-i}{2c_3}w\left(y\right)$
- $\int y^2\exp\left\{ic_3 y^2\right\}\,dy = \frac{-i}{4c_3^{3/2}}\left[\sqrt{i\pi}v\left(y\right) + 2\sqrt{c_3}y\,w\left(y\right)\right]$

and finally

$$Q_m = \mathrm{e}^{ic_1}\left[-\frac{d_1}{2}\sqrt{\frac{i\pi}{c_3}}v\left(y\right) - \frac{id_2}{2c_3}w\left(y\right) - \frac{id_3}{4c_3^{3/2}}\left[\sqrt{i\pi}v\left(y\right) + 2\sqrt{c_3}y\,w\left(y\right)\right]\right]. \tag{16}$$

3 Some examples

In his article Shampine introduces some examples to show the behavior of quadgF. Some of his examples are shown below, along with new ones.

It is very likely that his implementation of the function, `quadgF`, would be faster and more effective than mine. It is also possible that the `quadgk` function has improved since the publication of his article in November 2012. But the fact is that most of the examples that include reference [1] are resolved perfectly by the standard `quadgk` function, with better accuracy and in less time than my implementation of the `quadgF` function. Maybe this would explain why the function quadgF has dropped off the radar.

However, these same examples, slightly modified, are no longer solved by quadgk with the default number of intervals and tolerances, but by quadgF. On the other hand, it is possible to use a larger number of intervals and larger tolerances with quadgk and obtain a good result. As it can see in the following examples, sometimes `quadgF` is faster than `quadgk`, but not always.

3.1 Example 1

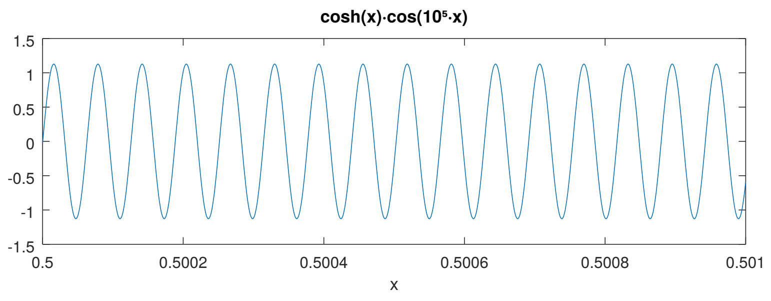
The first Shampine's example is:

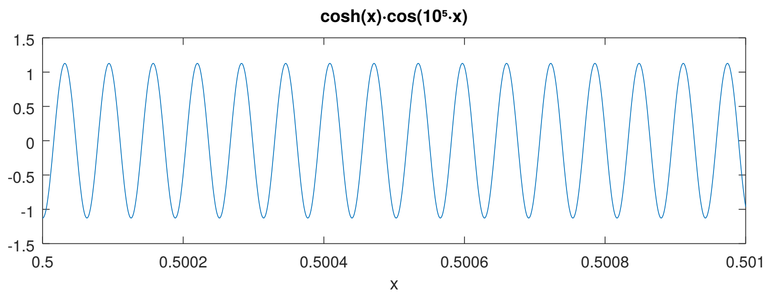
$$I = \int_0^1 \cosh(x) \exp\left\{i\, 10^5 x\right\}\, dx\,.$$

This integrant can be integrated analitically, so we can obtain the exact solution:

$$I = \frac{e^{i10^5 x}}{10^{10}+1}\left[\sinh(x) - i10^5\cosh(x)\right]\Bigg]_0^1 = 5.5152\, 10^{-7} + i\, 2.5421\, 10^{-5}$$

The following graph shows the integrant behavior in the integration interval. The function is plotted only on a small sub-interval, $x \in [0.5, 0.501]$, because if the complete interval was represented, a blur would be seen occupying the whole graph, since the integrant oscillates very strongly.





- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) cosh(x), @(x) 1e5*x, 0, 1);`
    - qa = 5.5152e-07 + 2.5421e-05i
    - num_iter = 1
    - error = 9.6e-15 - i 2.3e-16
    - average run time = 1.4 ms

- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) cosh(x).*exp(i*1e5*x),0,1);`
    - warning: quadgk: maximum interval count (650) exceeded
    - qb = 0.0011805 + 0.0024810i
    - err = 0.34280 (Error estimated by quadgk)
    - err = 1.2e-3 + i 2.5e-3 (Error obtained by comparing with the exact result)
- It is obvious that in order to solve this integral, quadgk needs to use more intervals, or a bigger tolerance. Let us use more intervals:
  - `[qb, err] = quadgk(@(x) cosh(x).*exp(i*1e5*x),0,1,"MaxIntervalCount",100000);`
    - qb = 5.5152e-07 + 2.5421e-05i
    - err = 6.4e-11 (Error estimated by quadgk)
    - error = 8.4e-15 - i 5.3e-15 (Error obtained by comparing with the exact result)
    - average run time = 0.12 s

3.2 Example 2

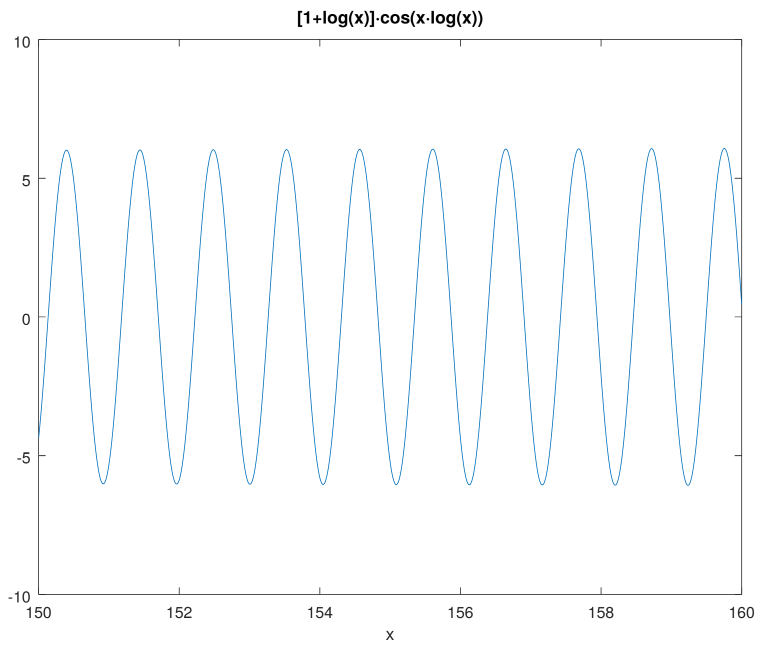
The second Shampine's example is:

$$I = \int_{100}^{200} \left[1 + \log(x)\right] \cos\left(x \log(x)\right) \, dx$$

This integrant can also be integrated analytically, so we can obtain the exact solution:

$$I = \sin\left(x \log(x)\right)\Big]_{100}^{200} = -1.77429897490598$$

The following graph shows the integrant behavior in the integration interval. In order to make the figure clearer, the graph is plotted only on the sub-interval $x \in [150, 160]$. As it can be seem, the integrant oscilates, but less strongly than the previous example.



- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) 1+log(x), @(x) x.*log(x), 0, 1);`
    - real(qa) = -1.77428424824785
    - num_iter = 1

    - error = 1.5e-05;
    - average run time = 1.5 ms

- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) (1+log(x)).*cos(x.*log(x)),100,200);`
    - qb = -1.77429897490590
    - err = 8.8e-07 (Error estimated by quadgk)
    - err = 7.8e-14 (Error obtained by comparing with the exact result)
    - average run time = 2.4 ms

As can be seen, `quadgF` does not always obtain better results than `quadgk`, even with strongly oscillating integrants.

3.2.1 A little variation

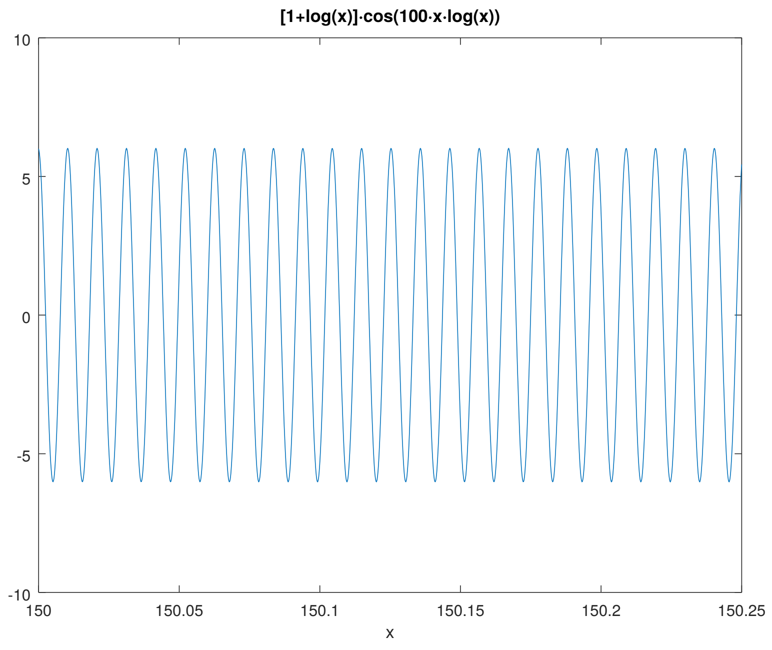
Let us consider an integrant slightly different from Shampine's second example:

$$I = \int_{100}^{200} [1 + \log(x)] \cos(100\, x\, \log(x))\, dx$$

whose analytical solution is:

$$\left. \frac{\sin(100\, x\, \log(x))}{100} \right]_{100}^{200} = -0.00372073265050140$$

The following graph shows the integrant behavior in the integration interval. For clarity the graph is plotted only on the sub-interval, $x \in [150, 150.25]$. As can be seen the integrant oscillates quite strongly.



- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) 1+log(x), @(x) x.*log(x), 0, 1);`
    - real(qa) = -0.00372073265050140
    - num_iter = 2
    - error = 2.5e-08;
    - average run time = 6.6 ms

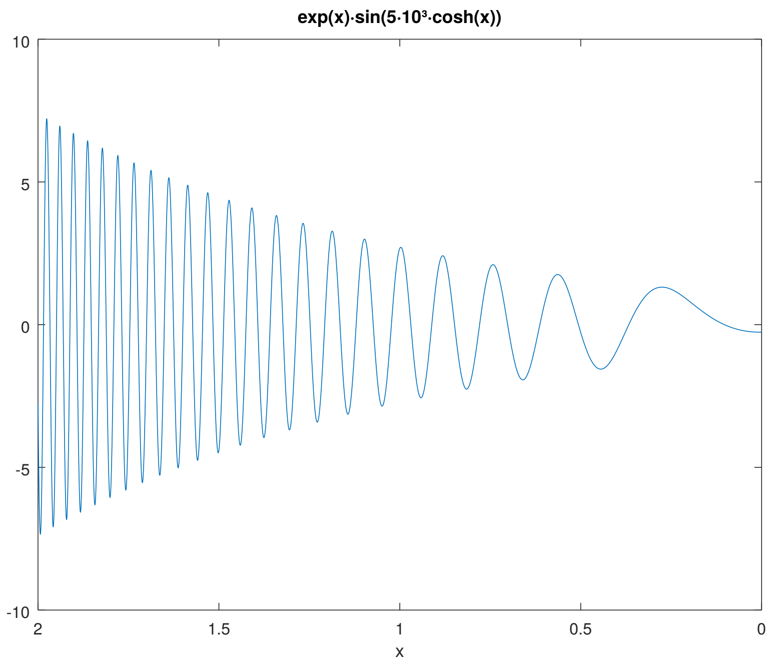
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) (1+log(x)).*cos(x.*log(x)),100,200);`
    - warning: quadgk: maximum interval count (650) exceeded
    - qb = -7.16438876031491
    - err = 79.1 (Error estimated by quadgk)
    - err = 7.2 (Error obtained by comparing with the exact result)
- Let us use more intervals and a bigger tolerance:
  - `[qb, err] = quadgk(@(x) (1+log(x)).*cos(x.*log(x)), 100, 200, "AbsTol", 1e-7, "MaxIntervalC`
    `100000);`
    - qb = -0.00372075782559319
    - err = 5.6e-08 (Error estimated by quadgk)
    - error = 1.2e-12 (Error obtained by comparing with the exact result)
    - average run time = 0.54 s

3.3 Example 3

The third Shampine's example is:

$$\int_{2}^{0} e^{x} \sin\left(50 \cosh\left(x\right)\right) dx$$

This integral has not analytical solution. The integrant behavior in the integration interval is shown in the following graph.



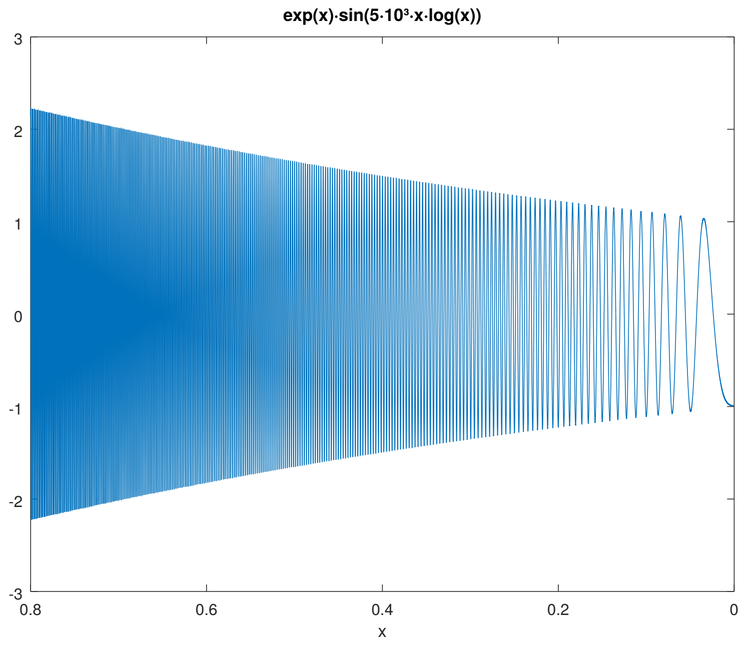
- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) exp(x), @(x) 50*cosh(x), 2, 0);`
    - imag(qa) = -0.070765
    - num_iter = 3
    - average run time = 6.6 ms
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) exp(x).*sin(50*cosh(x)),2,0);`
    - qb = -0.070765
    - err = 8.1e-09 (Error estimated by quadgk)
    - average run time = 1.8 ms

3.3.1 A little variation

Let us consider an integrant slightly different from Shampine's third example:

$$\int_2^0 \mathrm{e}^x \sin\left(5\,10^3 \cosh(x)\right)\, dx$$

The following graph shows the integrant behavior in the integration interval. For clarity the graph is plotted only on the sub-interval, $x \in [0.8, 0]$. As can be seen the integrant oscillates very strongly.



- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) exp(x), @(x) 5e3*cosh(x), 2, 0);`
    - imag(qa) = 0.0106709818154785
    - num_iter = 4
    - average run time = 23.7 ms
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) exp(x).*sin(5e3*cosh(x)),2,0);`
    - warning: quadgk: maximum interval count (650) exceeded
    - qb = 0.0130227181068779
    - err = 1.1 (Error estimated by quadgk)
- Let us use more intervals:
  - `[qb, err] = quadgk(@(x) exp(x).*sin(5e3*cosh(x)), 2, 0, "MaxIntervalCount", 10000);`
    - qb = 0.0106719656747756
    - err = 9.9e-09 (Error estimated by quadgk)
    - average run time = 13.7 ms

3.4 Example 4

The fourth Shampine's example

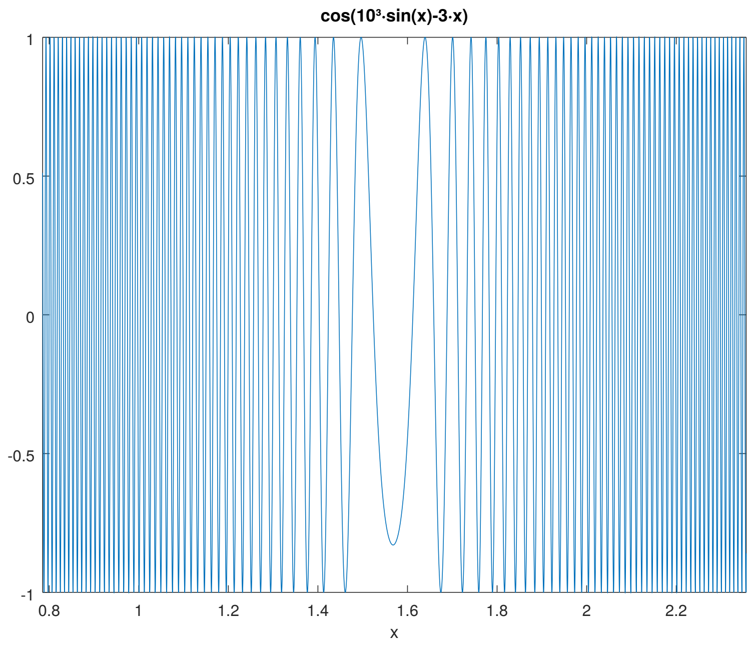
$$I = \int_0^\pi \cos\left(1000 \sin(x) - 3x\right)\, dx$$

has a first-order critical point at the point $\arccos\left(3\cdot 10^{-3}\right) = 1.56779632229488$, but this information is not given to the quadrature functions.

In this case $f(x) = 1 = cte$. This kind of functions is easily included in a octave script as:

```
f = @(x) cte*ones(size(x));
```

The following graph shows the integrant behavior in the integration interval. For clarity the graph is plotted only on the sub-interval, $x \in \left[\frac{\pi}{4}, \frac{3\pi}{4}\right]$. As can be seen the integrant oscillates very strongly.



- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) ones(size(x)), @(x) 1e3*sin(x)-3*x, 0, π,1e-8,10);`
    - real(qa) = -0.0151657899919954
    - num_iter = 6
    - error = 1e-09;
    - average run time = 0.15 s
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) cos(1e3*sin(x)-3*x),0,π)`
    - qb = -0.0151657898002485
    - err = 1e-09 (Error estimated by quadgk)
    - average run time = 3.7 ms

As can be seen, `quadgF` does not always obtain better results than `quadgk`, even with strongly oscillating integrants.

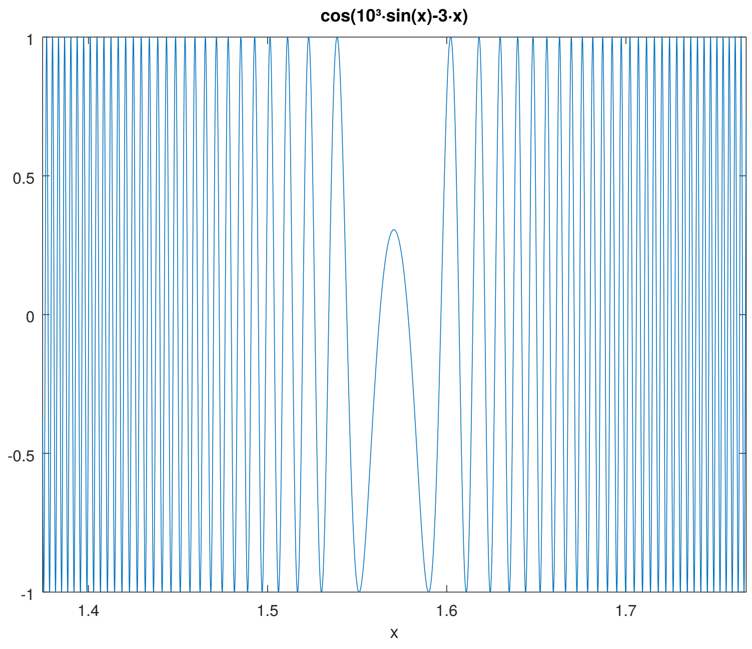
3.4.1 A little variation

Let us consider an integrant slightly different from Shampine's fourth example:

$$I = \int_0^\pi \cos\left(10000 \sin(x) - 3x\right)\, dx\,.$$

There is a first-order critical point at the point $\arccos\left(3\cdot 10^{-4}\right) \approx \pi/2$, but this information is not given to the quadrature functions.

The following graph shows the integrant behavior in the integration interval. For clarity the graph is plotted only on the sub-interval, $x \in \left[\frac{7\pi}{16}, \frac{9\pi}{16}\right]$. As can be seen the integrant oscillates very strongly.



- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) ones(size(x)), @(x) 1e4*sin(x)-3*x, 0, π, 1e-8, 10);`
    - real(qa) = -0.0114498877548738
    - num_iter = 7
    - average run time = 0.27 s
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) cos(1e3*sin(x)-3*x),0,π,"MaxIntervalCount",10000)`
    - qb = -0.0114498862831192
    - err = 3.8e-09 (Error estimated by quadgk)
    - average run time = 0.63 s

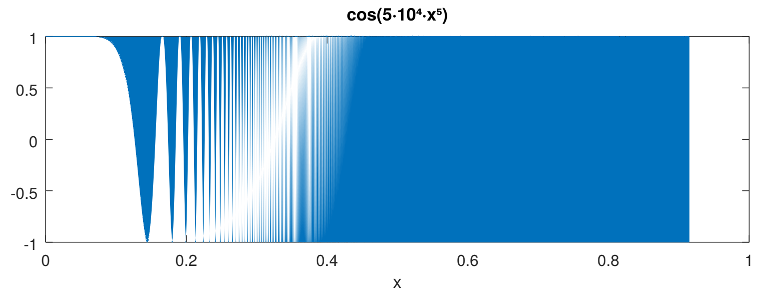
3.5 Example 5

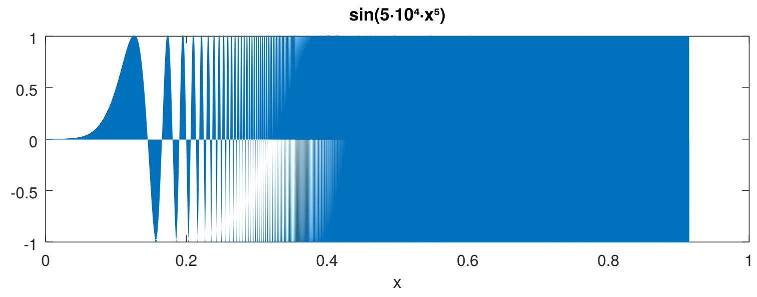
Now we consider a variation of the last Shampine's is:

$$\int_0^1 \exp\left\{i\,5000\,x^5\right\}\,dx$$

This integral can be expressed in terms of the incomplete Gamma function

$$I = -\left.\frac{\Gamma\left(\frac{1}{10}, -i\,5000x^5\right)}{\sqrt[10]{-i\,5000}}\right]_0^1 = 0.504642 + i\,0.0801191$$





- The numerical result that gives us the quadF function is:
  - `[qa, num_iter] = quadgF(@(x) ones(size(x)), @(x) 5e4*x.^5,0,1,1e-8,10);`
    - ∗ real(qa) = 0.1003038290338300 + i 0.0325920589819102
    - ∗ num_iter = 8
    - ∗ average run time = 0.36 s
- We can compare with the result of the standard quadgk function
  - `[qb, err] = quadgk(@(x) exp(i*5e4*x.^5),0,1,"MaxIntervalCount",10000);`
    - ∗ qb = 0.1003038290807608 + 0.0325920607196434i
    - ∗ err = 1.8e-08
    - ∗ average run time = 44 ms

As can be seen, `quadgF` does not always obtain better results than `quadgk`, even with strongly oscillating integrants.

References

[1] Shampine L.F., Integrating oscillatory functions in MATLAB, II. *Electronic Transactions on Numerical Analysis*, volume **39**, pp 403-413, 2012.

[2] Stamnes J.J., Spjelkavik B. & Pedersen H.M., Evaluation of diffraction integrals using local phase and amplitude approximations. *Opt. Acta* **30**, 207-222, 1983.

[3] Stamnes J.J., *Waves in focal regions: propagation, diffraction and focusing of light, sound and water waves.* Adam Hilger, 1986.