An Adaptive Contour Code for the Numerical Evaluation of the Oscillatory Cuspoid Canonical Integrals and their Derivatives

N.P. Kirk[a] J.N.L. Connor[b] C.A. Hobbs[c]

[a]*Department of Mathematical Sciences, University of Liverpool, Liverpool L69 3BX, England.*

[b]*Department of Chemistry, University of Manchester, Manchester M13 9PL, England.*

[c]*School of Computing and Mathematical Sciences, Oxford Brookes University, Gipsy Lane, Headington, Oxford OX3 0BP, England.*

Abstract

We present a code to compute the oscillatory cuspoid canonical integrals and their first order partial derivatives. The algorithm is based on the method of Connor and Curtis [*J.Phys.A*, **15** (1982) 1179–1190], in which the integration path along the real axis is replaced by a more convenient contour in the complex plane, rendering the oscillatory integrand more amenable to numerical quadrature. Our code is a modern implementation of this method, presented in a modular fashion as a FORTRAN 90 module containing the relevant subroutines. The code is robust, with the novel feature that the algorithm implements an adaptive contour procedure, choosing contours that avoid the violent oscillatory and exponential natures of the integrand and modifying its choice as necessary. In many cases, the subroutines are efficient and provide results of high accuracy. Plots and results for the Pearcey and swallowtail cuspoid integrals given in previous articles are reproduced and verified. Our subroutines significantly extend the range of application of this earlier work.

PROGRAM SUMMARY

Title of package: cuspint.

Catalogue identifier: ...

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland.

Computers on which the program has been tested: Sun Sparc, SGI Indy, SGI Challenge.

Operating systems or monitors under which the program has been tested: Unix.

Programming language used: Fortran 90.

Distribution format: uuencoded compressed tar file.

Program Library subprograms used: NAG Program Library (subroutine D01AKF) or QUADPACK Program Library (subroutine DQAG and dependencies).

Keywords: cuspoid oscillatory integrals, Pearcey canonical integral, swallowtail canonical integral, adaptive contour numerical quadrature.

Nature of physical problem: The theoretical treatment of short wavelength scattering phenomena often involves the uniform asymptotic evaluation of oscillating integrals with several coalescing saddle points. An important practical problem then is the numerical evaluation of the cuspoid canonical integrals and their first order partial derivatives.

Method of solution: A robust Fortran 90 code has been developed in which the integration path along the real axis is replaced by a contour in the complex plane, one which is more suitable for a numerical quadrature [1]. Results for the cusp and swallowtail canonical integrals are presented.

Typical running time: The code takes a few seconds on a Sun Sparc Ultra 2 Workstation to compute the cusp (Pearcey) canonical integral, $P(x, y)$, and its derivatives, $\partial P(x, y)/\partial x$ and $\partial P(x, y)/\partial y$, on the grid $x = -8.0$ (2.0) 8.0 and $y = 0.0$ (2.0) 8.0; see Section 5.

Unusual features of the program: The code has the novel feature that the algorithm implements an adaptive contour procedure, which chooses contours that avoid the violent oscillatory and exponential natures of the integrand. Modifications to the contour are made by exploiting the powerful error reporting facilities of subroutine D01AKF in the NAG Program Library or subroutine DQAG and dependencies in the QUADPACK Program Library.

Reference:

[1] J.N.L. Connor, P.R. Curtis, A method for the numerical evaluation of the oscillatory integrals associated with the cuspoid catastrophes: application to Pearcey's integral and its derivatives, *J. Phys. A*, **15** (1982) 1179–1190.

1 Introduction

Asymptotic techniques play an important role in the theory of many short wavelength scattering phenomena. For example, in the scattering of acoustic, electromagnetic, geophysical and water waves as well as the collisions of atoms, molecules and nuclear ions, see e.g. [1–8] which give many references to earlier research.

Many of these theories involve the uniform asymptotic evaluation of oscillating integrals with several coalescing saddle points [1,2,9–14]. An important practical problem is then the numerical evaluation of certain *canonical integrals*. Of particular interest are the *cuspoid canonical integrals* [11].

$$C_n(\boldsymbol{a}) = \int_{-\infty}^{\infty} \exp[\mathrm{i} f_n(\boldsymbol{a};u)]\,\mathrm{d}u, \qquad \boldsymbol{a} = (a_1, a_2, \ldots, a_{n-2}) \tag{1}$$

and their $n-2$ first order partial derivatives

$$\frac{\partial C_n(\boldsymbol{a})}{\partial a_1}, \frac{\partial C_n(\boldsymbol{a})}{\partial a_2}, \ldots, \frac{\partial C_n(\boldsymbol{a})}{\partial a_{n-2}} \tag{2}$$

where

$$f_n(\boldsymbol{a};u) = u^n + \sum_{k=1}^{n-2} a_k u^k \tag{3}$$

with u and the a_k real, and n an integer greater than 2. The existence of the $C_n(\boldsymbol{a})$ and their partial derivatives (2), obtained by differentiating under the integral sign, is guaranteed by a theorem of Hardy [15].

The name of the $C_n(\boldsymbol{a})$ integral derives from the fact that $f_n(\boldsymbol{a};u)$ is a miniversal unfolding of a cuspoid singularity (also called an A_{n-1} singularity); see [16,17]. For $n = 3$, the integral is the well-characterised Airy integral, or fold canonical integral in the language of catastrophe theory. For $n = 4$, we have the Pearcey integral, or cusp canonical integral. The first practical application of the uniform asymptotic Pearcey approximation was to the theory of cusped rainbows in elastic $\mathrm{He}^+ + \mathrm{Ne}$ scattering [18] (see also [19]). The case $n = 5$ is the swallowtail canonical integral, which represents the limit in most works to date. The first applications of the uniform asymptotic swallowtail approximation were to the calculation of product vibrational distributions in the $H + F_2 \rightarrow HF + F$ chemical laser reaction [20] and to the asymptotics of the butterfly canonical integral [21].

The purpose of the present paper is to describe an efficient numerical method for the evaluation of $C_n(\boldsymbol{a})$ and its derivatives, and to present a code that implements this method. Observe that the integrand of equation (1) is infinitely oscillating along the real axis, rendering direct numerical evaluation impossible. Several numerical methods and their merits to overcome this problem have been discussed in a survey article by one of us [1] and in an article on the uniform asymptotic swallowtail approximation [21]. In summary, there is *first* a contour integral method that is efficient, gives high-accuracy results and is relatively easy to implement on a computer. This method applies to $C_n(\boldsymbol{a})$ and its partial derivatives and can be generalised to other types of oscillatory integrals such as the hyperbolic umbilic canonical integral [2]. A *second* method derives a set of differential equations to which $C_n(\boldsymbol{a})$ is a solution and then solves these equations numerically [18,22–24]. This method has several advantages: the derivatives are obtained automatically, at the same time as $C_n(\boldsymbol{a})$. In addition, a step-by-step numerical technique for solving the differential equations provides an efficient way of generating grids for use in plotting. Its disadvantages include: derivation of the differential equations and their initial conditions is non-trivial; the method is difficult to implement on a computer, especially if one wants a subroutine to handle the case of general n; for certain values of $\boldsymbol{a}$ the independent solutions of the differential equations are exponentially increasing, thereby limiting the accuracy to which $C_n(\boldsymbol{a})$ can be calculated. Finally we mention a *third* method: the use of series representations for $C_n(\boldsymbol{a})$. These are important since they converge for all values of $\boldsymbol{a}$ and are a useful aid for checking results from new code. They can be used when the values of the coefficients a_k are small but, in general, are too inefficient for large a_k values. Previous work and experience suggests that the contour method is the most practical.

There has long been a need for robust efficient subroutines to calculate the cuspoid canonical integrals and their derivatives. Indeed one of us (JNLC) is often asked about the programs developed in his earlier work. These programs, which were developed in the early 1980s, represented a significant achievement for the computation of $C_n(\boldsymbol{a})$. However, they are no longer available and it is clear that a modern implementation, which extends the functionality of the older subroutines, is essential.

The code presented in this paper is an extension of the contour integral method. In particular, it implements an *adaptive contour algorithm* that makes the calculation more robust. It is written in a modular fashion using FORTRAN 90. It should be noted that integration along complex contours does not figure prominently in treatises on numerical integration [25–27] as a method for handling infinitely oscillating integrals. However, we shall demonstrate that exploiting complex integration paths is easy to implement and capable of high accuracy (see also [28–31]).

We next explain our choice of program library quadrature subroutines. As will be shown in Section 2, it is not possible to completely avoid the oscillatory nature of the integrand. Rather we choose a contour that provides a compromise between the violent oscillatory and exponential natures of the integrand in the complex u plane. Since we wish to provide an efficient and high-precision code it is desirable to use specialist quadrature subroutines, specifically suited to oscillating, non-singular integrands. To this end, we have written two versions of the code in FORTRAN 90 and the user must choose the more suitable implementation. The first version uses the subroutine D01AKF present in the NAG Program Library [32]; the second version uses the subroutine DQAG and dependencies present in the publicly available QUADPACK Program Library [33]. The subroutine D01AKF, which is based on DQAG, is an adaptive integrator especially suited to oscillating non-singular integrands over a finite interval. It uses the Gauss 30 point and Kronrod 61 point rules with a global acceptance criterion as defined by Malcolm and Simpson [34]. A further advantage is that these subroutines have powerful error detection facilities. Our algorithm acts on errors returned by the quadrature subroutines, modifying the contour of integration accordingly, which results in the adaptive contour algorithm mentioned above.

The contour integral method is explained in Section 2, whilst our implementation of it is described in Section 3. The specifications of the computer codes in our package are given in Section 4. Test inputs and outputs are discussed in Section 5.

2 Contour Integral Method

We first present a summary of the method described in reference [35]. We begin by writing equation (1) in the form

$$C_n(\boldsymbol{a}) = \int_0^\infty \exp[\mathrm{i}f_n(\boldsymbol{a};u)]\,\mathrm{d}u + \int_0^\infty \exp[\mathrm{i}f_n(\boldsymbol{a};-u)]\,\mathrm{d}u. \tag{4}$$

Denote these integrals by $I_n^+(\boldsymbol{a})$ and $I_n^-(\boldsymbol{a})$, respectively. [In general, we have to consider $I_n^+(\boldsymbol{a})$ and $I_n^-(\boldsymbol{a})$ separately, but for n an even integer it is only necessary to consider an integral of the type $I_n^+(\boldsymbol{a})$.] Now replace the contours of integration (the positive real axis) as follows. For $I_n^+(\boldsymbol{a})$ we use the ray from 0 to $\infty\exp(\mathrm{i}\pi/2n)$, while for $I_n^-(\boldsymbol{a})$ the ray from 0 to $\infty\exp[(-1)^n\mathrm{i}\pi/2n]$ is employed. These changes do not effect the values of the integrals. This is because each integrand is entire and an application of Cauchy's Theorem shows that we can replace the original contours along the real axis by the above rays together with arcs from $\infty \exp[(-1)^n \mathrm{i}\pi/2n]$ to ∞. Then, by an application of Jordan's Lemma, the contributions to the integrals from the arcs at infinity vanish and may be ignored.

Let $u_n^+(t)$ and $u_n^-(t)$ denote the natural parametrisations of the above rays: $u_n^+(t) = \exp(\mathrm{i}\pi/2n)t$ and $u_n^-(t) = \exp[(-1)^n \mathrm{i}\pi/2n]t$. Observe that on substituting $u = u_n^+(t)$ into $\mathrm{i}f_n(\boldsymbol{a}; u)$, and $u = u_n^-(t)$ into $\mathrm{i}f_n(\boldsymbol{a}; -u)$, the leading term of each becomes $-t^n$. Hence, the integrands of $I_n^\pm(\boldsymbol{a})$ are now exponentially decreasing as $\exp(-t^n)$ for $t \to \infty$, making numerical evaluation feasible.

We will need explicit expressions for these integrands, so we present them here. We have

$$\begin{aligned} I_n^+(\boldsymbol{a}) &= \int_0^\infty \exp\{\mathrm{i}f_n[\boldsymbol{a}; {u_n}^+(t)]\}\, ({u_n}^+)'(t)\, \mathrm{d}t \\ &= \int_0^\infty \exp[-h_n^+(\boldsymbol{a}; t)] \exp\left\{\mathrm{i}\left[\frac{\pi}{2n} + \sum_{k=1}^{n-2} \cos\left(\frac{\pi k}{2n}\right) a_k t^k\right]\right\} \mathrm{d}t, \end{aligned} \tag{5}$$

and

$$\begin{aligned} I_n^-(\boldsymbol{a}) &= \int_0^\infty \exp\{\mathrm{i}f_n[\boldsymbol{a}; -u_n^-(t)]\}\, (u_n^-)'(t)\, \mathrm{d}t \\ &= \int_0^\infty \exp[-h_n^-(\boldsymbol{a}; t)] \\ &\quad \times \quad \exp\left\{\mathrm{i}\left[\frac{(-1)^n\pi}{2n} + \sum_{k=1}^{n-2} (-1)^k \cos\left(\frac{\pi k}{2n}\right) a_k t^k\right]\right\} \mathrm{d}t, \end{aligned} \tag{6}$$

where

$$h_n^+(\boldsymbol{a}; t) = t^n + \sum_{k=1}^{n-2} \sin\left(\frac{\pi k}{2n}\right) a_k t^k, \tag{7}$$

$$h_n^-(\boldsymbol{a}; t) = t^n + (-1)^n \sum_{k=1}^{n-2} (-1)^k \sin\left(\frac{\pi k}{2n}\right) a_k t^k. \tag{8}$$

The bound

$$|I_n^\pm(\boldsymbol{a})| \le \int_0^\infty \exp[-h_n^\pm(\boldsymbol{a}; t)]\, \mathrm{d}t \tag{9}$$

****Contour Diagram here****

Fig. 1. Contours used for the numerical evaluation of $I_n^+(\boldsymbol{a})$.

together with the asymptotic behaviour, $h_n^\pm(\boldsymbol{a};t) \to t^n$ for $t \to \infty$, suggests that a numerical approximation to the $I_n^\pm(\boldsymbol{a})$ should be possible.

However, as observed in [35], there is still a problem. For certain values of the coefficients a_k, the $h_n^\pm(\boldsymbol{a};t)$ can possess large oscillations (in principle they have up to n real roots), which are magnified further by the exponential functions. Thus, the integrands may behave in a violent exponential manner before the leading terms in the exponents ($-t^n$ in both cases) become dominant. This problem should not be underestimated; it turns out to be as great an obstruction to the accurate numerical evaluation of the integrals as are the infinite oscillations along the real axis.

To solve this problem, we make a compromise in our choice of contours. In particular we replace the rays by three contours $C_1^\pm$, $C_2^\pm$ and $C_3^\pm$. Thus, the $C_1^\pm$ proceed from the origin along the real axis to *breakpoints* $R_0^\pm$. The contours $C_2^\pm$ are typically straight lines (alternatively arcs can be used) from $R_0^\pm$ to the points $R_0^\pm \exp(\pm\mathrm{i}\pi/2n)$, where the $-$ sign in the exponential applies when n is an odd integer (*this convention will be used from now on*). The contours $C_3^\pm$ continue along the original rays, going from $R_0^\pm \exp(\pm\mathrm{i}\pi/2n)$ to $\infty \exp(\pm\mathrm{i}\pi/2n)$. This is shown for the $+$ case in Figure 1. The problem now reduces to choosing suitable values for the breakpoints $R_0^\pm$. The choices are discussed fully in Section 3, but the basic idea is to choose the $R_0^\pm$ large enough so that if $t \geq R_0^\pm$ then the dominant terms in $h_n^\pm(\boldsymbol{a};t)$ are t^n and the integrands are well behaved along the $C_3^\pm$. For example, if $U_0^\pm$ are upper bounds on the sets of real roots of $h_n^\pm(\boldsymbol{a};t)$, then if $t \geq U_0^\pm$ we have $h_n^\pm(\boldsymbol{a};t) \geq 0$ and $\exp[-h_n^\pm(\boldsymbol{a};t)] \leq 1$.

The trade off with this new choice of contours is that along $C_1^\pm$ the integrands are oscillatory, this being the main obstruction to the accurate numerical evaluation of the $I_n^\pm(\boldsymbol{a})$ at the outset. However, with suitable choices for the $R_0^\pm$, the high-quality quadrature subroutines now available provide efficient and accurate approximations to the $I_n^\pm(\boldsymbol{a})$. We require choices for $R_0^\pm$ which ensure that the integrands are moderately well behaved along all six possible contours or, failing that, a scheme for modifying the $R_0^\pm$. The latter is possible owing to the robust error detection facilities in the quadrature subroutines that we use.

3 Algorithm for the Numerical Evaluation of the Cuspoid Integrals

Here we present our implementation of the contour integral method described in Section 2.

3.1 Adaptive Contour Method

We choose the $R_0^\pm$ to be upper bounds for the positive real roots of the polynomials $h_n^\pm(\boldsymbol{a};t) - t$ as given by the Cauchy bound described in Section 3.3 (1). [Note that $t = 0$ is always a root of the $h_n^\pm(\boldsymbol{a};t) - t$.] This not only ensures that the $h_n^\pm(\boldsymbol{a};t)$ are well-behaved along the $C_3^\pm$, but it provides us with specific bounds for the corresponding integrals. Thus, if $t \geq R_0^\pm$ then $h_n^\pm(\boldsymbol{a};t) \geq t$ so that $\exp[-h_n^\pm(\boldsymbol{a};t)] \leq \exp(-t)$. Given some $M^\pm \geq R_0^\pm$, the errors in approximating the integrals along the infinite rays $C_3^\pm$ by integrals along *finite* rays from $R_0^\pm \exp(\pm \mathrm{i}\pi/2n)$ to $M^\pm \exp(\pm \mathrm{i}\pi/2n)$ are given by

$$\int_{M^\pm}^{\infty} \exp[-h_n^\pm(\boldsymbol{a};t)]\,\mathrm{d}t \leq \int_{M^\pm}^{\infty} \exp(-t)\,\mathrm{d}t = \exp(-M^\pm). \tag{10}$$

Given an error tolerance T, we next set

$$M^\pm = \max(R_0^\pm, -\ln T). \tag{11}$$

Then, if $M^\pm = -\ln T$, it is necessary to perform a quadrature for the integrals along $C_3^\pm$ from $R_0^\pm \exp(\pm \mathrm{i}\pi/2n)$ to $M^\pm \exp(\pm \mathrm{i}\pi/2n)$, whereas if $M^\pm = R_0^\pm$, the integrals along $C_3^\pm$ are negligible relative to the error tolerance. Figure 1 shows a typical location of $M^\pm \exp(\pm i\pi/2n)$ for the + case.

Choosing large values for $R_0^\pm$ ensures that the integrands along $C_3^\pm$ are well-behaved (or even negligibly small). A disadvantage is that these large values make the evaluation of the integrals along $C_1^\pm$ more difficult. To mitigate this problem, we can 'cut the corner' and break away from the real axis at points $R^\pm$ closer to the origin, thereby restricting the range of the violent oscillations for $C_1^\pm$. In particular, we use straight line segments from $R^\pm$ to $R_0^\pm \exp(\pm \mathrm{i}\pi/2n)$ in place of $C_2^\pm$; these contours are labelled $C_2'^\pm$ and are illustrated in Figure 1 for the + case.

We now summarise our adaptive contour algorithm for the calculation of the integrals $I_n^\pm(\boldsymbol{a})$. An iterative process is used which modifies the choice of $R^\pm$ depending on the success, or otherwise, of the quadratures along $C_1^\pm$ and $C_2'^\pm$. The iterates of $R^\pm$ are chosen to lie in intervals $[R_{\mathrm{inf}}^\pm, R_{\mathrm{sup}}^\pm]$, which are also refined (narrowed) during the iteration, resulting in a sequence of values for $R^{\pm}$ which converge to (and ideally attain) suitable breakpoints. More precisely, for each integral $I_n^+(\boldsymbol{a})$ and $I_n^-(\boldsymbol{a})$, the following algorithm is carried out. For simplicity we describe the + case; the − case is essentially the same (in fact, the code deals with the real and imaginary parts separately, although this technicality is not important to the present discussion).

(a) Initialise the iteration, setting $R^+ = R_0^+$ and $[R_{\text{inf}}^+, R_{\text{sup}}^+] = [0, R_0^+]$.

(b) The following steps are then repeated a predefined number of times (the default is 5) until the calculation succeeds, or an error condition arises (typically this occurs before the default is attained):

- Calculate the integrals along the contours C_1^+ and $C_2'^+$ using the current value of R^+.
- If the quadrature along C_1^+ fails, the problem lies with the oscillatory nature of the integrand along this contour and R^+ should be iterated towards R_{inf}^+. Thus, first check to see if the quadrature along $C_2'^+$ has also failed (as one cannot then calculate the next value for R^+ in the iteration) and return an error if necessary. Otherwise define the next iterate for the interval $[R_{\text{inf}}^+, R_{\text{sup}}^+]$ to be $[R_{\text{inf}}^+, R^+]$, that is set $R_{\text{sup}}^+ := R^+$, where ':=' denotes the assignment operator. *After* this assignment we define the next iterate for R^+ via $R^+ := (R_{\text{inf}}^+ + R^+)/2$.
- If the quadrature along $C_2'^+$ fails, the problem lies with the exponential nature of the integrand along this contour and R^+ should be iterated toward R_{sup}^+. Thus, replace $[R_{\text{inf}}^+, R_{\text{sup}}^+]$ by $[R^+, R_{\text{sup}}^+]$ by setting $R_{\text{inf}}^+ := R^+$. *After* this assignment we set $R^+ := (R^+ + R_{\text{sup}}^+)/2$.
- If neither quadrature fails, then an exit is made from the loop.

(c) Finally the integral along the truncated contour C_3^+ is calculated. The quadrature subroutines should perform this computation without any problems. (However, as a safety check their error return flags must be examined. Problems may arise if the user requests unreasonable error tolerances, for example.) More specifically: if $R_0^+ < M^+$ then calculate the integral along C_3^+ and return an error message if unsuccessful. Otherwise, if $R_0^+ = M^+$, the integral is negligible for the required error tolerance and may be ignored.

Technical Note: in the program code, the above algorithm is carried out using an internal subroutine named **calculate_I**, which is called four times by its host **cuspint_e**, with each call calculating a component $I_n^{\pm}(\boldsymbol{a})$. More specifically, the four components are given by a choice of +/− type and real/imaginary part, this being indicated by the arguments present in the call. A full breakdown of the calculation, giving the values of these components of $I_n^{\pm}(\boldsymbol{a})$ along each of the six optimised contours $C_1^{\pm}$, $C_2'^{\pm}$ and $C_3^{\pm}$, is returned.

Each of the twelve quadratures gives results whose sum yields an approximation Q to the actual integral $C_n(\boldsymbol{a})$. The adaptive contour subroutine returns Q, a bound for the absolute error $|C_n(\boldsymbol{a}) - Q|$ and an error flag to report on the success, or otherwise, of the calculation. If required, the subroutine also returns results from the individual quadratures, their error bounds and error flags, the Cauchy bounds $R_0^{\pm}$, and the values for $R^{\pm}$ produced in the iteration.

3.2 Error Bounds and Tolerances

The error tolerance $\varepsilon > 0$ and the choice of error bound (absolute or relative) may be specified by the user in the calling sequence. (The defaults are: $\varepsilon = 10^{-5}$ and absolute error.) Suppose Q denotes the numerical approximation to $C_n(\boldsymbol{a})$, then we have the following

- absolute case: $|C_n(\boldsymbol{a}) - Q| \leq \varepsilon$,
- relative case: $|C_n(\boldsymbol{a}) - Q| \leq \varepsilon |C_n(\boldsymbol{a})|$,

where $|\cdot|$ denotes the modulus of a complex number.

Our algorithm cannot guarantee, but in practice usually achieves, the following accuracy for the two cases:

3.2.1 Absolute Error Case

Errors are introduced during the twelve quadratures needed to calculate $I_n^{\pm}(\boldsymbol{a})$ (specifically: the calculation of the real and imaginary parts of $I_n^{\pm}(\boldsymbol{a})$ along the six contours $C_1^{\pm}$, $C_2'^{\pm}$ and $C_3^{\pm}$). Denote the real and imaginary parts of these errors by $\varepsilon_j = \varepsilon_j^{(r)} + \mathrm{i}\varepsilon_j^{(i)}$, where $j = 1, \ldots, 6$. Errors are also introduced by truncating the infinite contours $C_3^{\pm}$ for $I_n^{\pm}(\boldsymbol{a})$; let the two corresponding errors be denoted $\varepsilon_\infty^{\pm}$ (these represent complex numbers). Then the total absolute error in the approximation is given by

$$\left| \sum_{j=1}^{6} \varepsilon_j^{(r)} + \mathrm{i} \sum_{j=1}^{6} \varepsilon_j^{(i)} + \varepsilon_\infty^{+} + \varepsilon_\infty^{-} \right| \leq \sum_{j=1}^{6} |\varepsilon_j^{(r)}| + \sum_{j=1}^{6} |\varepsilon_j^{(i)}| + |\varepsilon_\infty^{+}| + |\varepsilon_\infty^{-}|. \quad (12)$$

For simplicity, we insist that each of the terms on the right hand side of relation (12) is less than $\varepsilon/20$, thus ensuring that the total absolute error is less than ε. For the first twelve error terms, this entails calling each quadrature subroutine with an absolute error tolerance of $\varepsilon/20$. The remaining two terms are dealt with by setting the upper limits of integration $M^{\pm} = \max[R_0^{\pm}, -\ln(\varepsilon/20)]$ in equation (11).

Finally, if $\delta_j^{(r)}$ and $\delta_j^{(i)}$ denote the corresponding *estimates* for the absolute errors, as returned by the twelve quadrature subroutines (under normal circumstances these are *bounds*), then a bound for the absolute error $|C_n(\boldsymbol{a}) - Q|$ is

$$\sum_{j=1}^{6} \delta_j^{(r)} + \sum_{j=1}^{6} \delta_j^{(i)} + \exp(-M^+) + \exp(-M^-). \tag{13}$$

3.2.2 Relative Error Case

It is a simple exercise with inequalities to show that one cannot guarantee a total relative error by enforcing relative errors on the individual components of the calculation (in contrast to the absolute case). Instead our algorithm assumes that $|[C_n(\boldsymbol{a}) - Q]/Q|$ is a good approximation to the relative error $|[C_n(\boldsymbol{a}) - Q]/C_n(\boldsymbol{a})|$; it then attempts to calculate an approximation Q satisfying

$$|C_n(\boldsymbol{a}) - Q| \le \varepsilon |Q|. \tag{14}$$

We start with $\varepsilon/2$ on the right hand side of criterion (14). That is, since no initial approximation Q is available, we assume an absolute tolerance of $\varepsilon/2$ to begin the iteration [observe that the left hand side of relation (14) is the absolute error]. Next, having *calculated* an approximation Q using our algorithm with this absolute error tolerance, we check to see if the relative error criterion (14) holds. [Note that we obtain a bound for the absolute error $|C_n(\boldsymbol{a}) - Q|$ from the bounds returned from the quadrature subroutines, as described above.] If the criterion fails then a new approximation Q is calculated from the current value, Q_{cur}, using an absolute error tolerance of $\varepsilon|Q_{cur}|$. This is repeated until the criterion (14) is satisfied or the maximum number of iterations, as specified by an optional argument, has been performed (the default for the number of iterations is 5).

3.3 Remarks on the Adaptive Contour Algorithm

(1) We calculate the upper bounds $R_0^{\pm}$ by applying the following proposition, sometimes called the *Cauchy Bound* [36, p. 20, item 1.2.3], to $h_n^{\pm}(\boldsymbol{a}; t) - t$.

Proposition *Let $p(X) = X^n + a_{n-1}X^{n-1} + \cdots + a_1 X + a_0$ be a monic polynomial with real coefficients and let α be a real root of p. Let $i_1, \ldots, i_r$ be the indices of those a_i which are negative. Then*

$$\alpha \le \max\left\{0, (r|a_{i_1}|)^{\frac{1}{n-i_1}}, \ldots, (r|a_{i_r}|)^{\frac{1}{n-i_r}}\right\}.$$

Note that $t = 0$ is always a root of $h_n^{\pm}(\boldsymbol{a}; t) - t$. Our use of this proposition, and the initial choices $R^{\pm} = R_0^{\pm}$, are backed up by numerical experiments, proving to be both efficient and satisfactory for normal usage. Indeed, for the simpler test cases described in Section 5, the initial choices $R^{\pm} = R_0^{\pm}$ are sufficient for a successful computation; the iterative procedure to refine the choices of the $R^{\pm}$ is only needed for more extreme examples.

(2) In order to handle more difficult cases, we allow the user to specify the initial choices for the intervals $[R_{\text{inf}}^{\pm}, R_{\text{sup}}^{\pm}]$ via the optional arguments **r_interval_p** and **r_interval_m**. These are arrays, each having two elements, which locate the end points $R_{\text{inf}}^{\pm}$, $R_{\text{sup}}^{\pm}$ of the initial intervals; their subscripts refer to the "plus" and "minus" cases. The initial breakpoints $R^{\pm}$ are then set equal to the midpoints of the intervals.

However, we stress that the default setup works well for normal usage. The user need only override it in extreme cases, typically based upon results from previous calculations which have run into difficulties. For example, the bounds $R_0^{\pm}$ may be too large, resulting in bad initial values for $R^{\pm}$ and needless iterations. Another scenario occurs when the $R_0^{\pm}$ are too large for $C_1^{\pm}$ and the first iterate produces breakpoints $R^{\pm} = R_0^{\pm}/2$, for which the exponential behaviour of an integrand along $C_2'^{\pm}$ is so extreme that an arithmetic overflow occurs. By selecting better initial intervals for the iteration, the user can often avoid such problems.

Possible improvements to the subroutine include: the implementation of tighter bounds for the $R_0^{\pm}$ or the direct numerical evaluation of the largest real roots of the $h_n^{\pm}(\boldsymbol{a}, t) - t$ using efficient, high quality root finders. However, problems similar to those just described will eventually arise for extreme values of the parameter $\boldsymbol{a}$, which again will require some form of user intervention.

(3) Explicit expressions for the real and imaginary parts of the integrands along each of the six contours are provided by twelve module functions in the source code. Along $C_1^{\pm}$, the integrands are obtained from equation (4), and similarly along $C_3^{\pm}$ they are given by equations (5) and (6). Along $C_2'^{\pm}$, the simplest approach is to exploit the complex arithmetic built into FORTRAN 90 rather than to derive explicit expressions. Thus, the straight line contours from $R^{\pm}$ to $R_0^{\pm}\exp(\pm \mathrm{i}\pi/2n)$ are parametrised by $u_n^{\pm}(t) = R^{\pm} + [R_0^{\pm}\exp(\pm \mathrm{i}\pi/2n) - R^{\pm}]t$, for $0 \le t \le 1$, which are then substituted into the integrands of equation (4).

(4) An extra monomial factor of $\mathrm{i}u^j$ is present in the integrand of equation (1) when calculating the derivative $\partial C_n(\boldsymbol{a})/\partial a_j$ because

$$\frac{\partial C_n(\boldsymbol{a})}{\partial a_j} = \int_{-\infty}^{\infty} iu^j \exp[\mathrm{i}f_n(\boldsymbol{a}; u)]\,\mathrm{d}u, \qquad j = 1, 2, \ldots, n-2 \tag{15}$$

We use the same adaptive contour algorithm to evaluate the derivatives as employed for $C_n(\boldsymbol{a})$. Indeed the only extension is to include a function that returns the integrand of $\partial C_n(\boldsymbol{a})/\partial a_j$. This is accomplished using the functions described in point (3) above, with an extra flag to indicate whether to return the integrand of $C_n(\boldsymbol{a})$ or that of one of its derivatives.

(5) In [35], the $R_0^{\pm}$ were usually chosen sufficiently large in order to make the integrals along $C_3^{\pm}$ negligible. In the present work we use (possibly smaller) values for $R_0^{\pm}$ which ensure only that the integrands are well-behaved. We favour values for $R_0^{\pm}$ that are less likely to produce quadrature problems along the real axis, where the integrands are oscillatory, even if this means we have to perform quadratures for the integrals along $C_3^{\pm}$. Note that the integrals along $C_3^{\pm}$ are relatively easy to compute, whereas those along $C_1^{\pm}$ require a far more intensive computation.

(6) Another computer code for the numerical evaluation of the cuspoid canonical integrals and their derivatives has been reported by Lukin et al. [37] (see also [8,38]). It uses straight-line paths in the complex u plane supplemented by Taylor series expansions. Thus for $I_n^+(\boldsymbol{a})$, contours with $\arg u = \pi/4n$, $\pi/16n$, $\pi/64n$, $\pi/256n$ are employed. From amongst these contours, an optimal one is chosen using the criterion that the modulus of the integrand does not exceed 10^6. A finite quadrature is then performed using Simpson's Rule, with the results being accurate up to the 5th decimal place. This algorithm can be considered a special case of that described in [35], which allows for many different contours lying in the sector $0 < \arg u < \pi/n$. The present adaptive contour code significantly extends the range of application of these earlier codes.

4 Package Specification

The package contains a FORTRAN 90 module which provides the relevant procedures and constants; a specification is provided below. In addition, all relevant documentation and procedure specifications are contained within the program code, and the FORTRAN 90 source is fully commented. The package also contains driver programs demonstrating usage of the subroutines. Documentation for these programs is provided in the distribution and is not reproduced in this article. Full details may be found in the README file. Two versions of the code are available, one using the NAG Program library and the other using the QUADPACK Program library. We provide the specifications for the NAG version below. The specifications for the QUADPACK version differ only slightly and may be found in the distribution.

4.1 Module Specification

The module **cuspoid_integral** provides the following public entities:

The subroutine **cuspint** evaluates a cuspoid integral, while the subroutine **cuspint_e** is an extended version passing back more information. The constant **Rkind** specifies the real kind type to be used in all floating point declarations. The subroutines contain optional arguments which take default values if not present in the calling sequence. The default values for the arguments, which control issues such as accuracy and workload, are available as the public constants **err_eps_def**, **err_abs_def**, **lw_def**, **liw_def**, **r_iter_def** and **rel_iter_def** (these apply to the NAG version).

The modular programming (data hiding) paradigm is particularly suitable for our problem. For example, explicit expressions for the real and imaginary parts of an integrand along each of the six contours are required and these must be specified as *module functions* as they are passed to the quadrature subroutines. These functions are of no concern to the end-user of the package and are hidden in the module. Similarly, the error handling subroutines and the function which calculates $R_0^{\pm}$ are specified by the module as having private access. Since expressions for the integrands contain the external parameters $\boldsymbol{a} = (a_1, a_2, \ldots, a_{n-2})$, the module functions defining the real and imaginary parts of the integrands have a single argument (called t) which needs to access the a_i as global variables in order to evaluate a return value. However, these global a_i are hidden in the module from the end-user. Finally, we remark that encapsulation of all the entities in the module also provides the inbuilt advantages which come with the use of FORTRAN 90 modules.

4.2 Subroutine Specifications

The routine **cuspint_e** implements the adaptive contour algorithm described in Section 3, whereas the subroutine **cuspint** is essentially a driver for **cuspint_e**. The following specifications are taken from the documentation which comes with the package; more details can be found there.

SUBROUTINE cuspint(a, deriv, result, abserr, ierror, err_eps, err_abs, lw, liw, r_iter, rel_iter, out_unit)

Arguments and Error Flags

The subroutine **cuspint_e** uses the same arguments as **cuspint** so, for brevity, we refer to the specification given below. The arguments **err_eps**, **err_abs**, **lw**, **liw**, **r_iter** and **rel_iter** specify the accuracy and workload parameters required by the subroutine. Note that these arguments are optional and take default values if not present in the calling sequence.

SUBROUTINE cuspint_e(a, deriv, result, abserr, ierror, res_cpts, abs_cpts, ifail_cpts, r_values, err_eps, err_abs, lw, liw, r_iter, rel_iter, out_unit, r_interval_p, r_interval_m)

Arguments

a(1:): real, input
The values of $a_1, a_2, \ldots$ at which the cuspoid integral is to be evaluated. The size of this array determines which member of the cuspoid family is considered. Specifically, if the component of **a** with largest index is $n-2$ then the unfolding is that of an A_{n-1} singularity, $u^n + \sum_{k=1}^{n-2} \mathbf{a}(k)u^k$.

deriv: integer, input
A key which specifies whether to return the value of the cuspoid integral (**deriv** $= 0$) or its partial derivatives with respect to $\mathbf{a}(k)$ (**deriv** $= k$ where $1 \leq k \leq n-2$).

result(2): real, output
Returns the real and imaginary parts of $C_n(\boldsymbol{a})$.

abserr: real, output
Returns what will, in normal circumstances, be an upper bound for the absolute error $|C_n(\boldsymbol{a}) - Q|$.

ierror: integer, input/output
On entry, this contains a flag which specifies what action the subroutine should take if it detects an error. On return it contains a flag reporting the success or otherwise of the calculation. See below.

res_cpts(2,3,2): real, output
Returns the real and imaginary parts of the integrals along the six contours into which $C_n(\boldsymbol{a})$ is decomposed. The first index of **res_cpts** specifies the integral type: $I_n^+(\boldsymbol{a})$ (index = 1) and $I_n^-(\boldsymbol{a})$ (index = 2); the second index specifies the contour (1, 2 or 3); and the third index specifies the real (1) and imaginary (2) parts.

abs_cpts(2,3,2): real, output
Returns the corresponding absolute error estimates for the real and imaginary parts of the integrals along the six contours. These are the error estimates returned by NAG subroutine D01AKF for the twelve component quadratures of the calculation and should be upper bounds; see [32].

ifail_cpts(2,3,2): integer, output
Returns the corresponding ifail flags, as specified by NAG subroutine D01AKF, for the real and imaginary parts of the integrals along the six contours; see [32].

r_values(2,2,0:r_iter): real, output
Returns the upper bounds $R_0^\pm$ and the iterates of $R^\pm$. The first index specifies the integral type: $I_n^+(\boldsymbol{a})/I_n^-(\boldsymbol{a})$; the second index specifies the

real/imaginary part; and the third index specifies the values of $R_0^{\pm}$ (index = 0) and iterates of $R^{\pm}$ (index = 1:**r_iter**) (the value -1.0 is used to pad out the array in the entries for unused iterates).

The following nine arguments specify the accuracy and workload parameters required by the subroutine. All arguments are optional and take default values if not present in the calling sequence.

err_eps: real, optional, input
Error tolerance. The default value is 10^{-5}.
Constraint: **err_eps** $\geq$ smallest possible real of kind **Rkind**; this is given by the intrinsic function 'tiny'.

err_abs: logical, optional, input
Use absolute error (**true**) or relative error (**false**). The default is **true**.

lw, liw: integer, optional, input
Dimensions for the work arrays in NAG subroutine D01AKF; see [32]. NAG suggested values: **lw** in the range 800 to 2000 and **liw** = **lw**/4. For calculating cuspoid integrals, the default values **lw** = 800, **liw** = 200 appear to work well. Taking **lw** larger may help for more extreme cases but, in general, this may make the NAG subroutine work harder than is necessary when calculating the integrals along $C_1^{\pm}$, instead of modifying the breakpoints. This has a detrimental effect on the overall efficiency of the subroutine.
Constraints: **liw** ≥ 1, **lw** $\geq 4\,$**liw**.

r_iter: integer, optional, input
Maximum number of iterations for the breakpoints $R^{\pm}$, while adapting the contours in order to achieve a successful computation. The default is 5.
Constraint: **r_iter** > 0.

rel_iter: integer, optional, input
Maximum number of times $C_n(\boldsymbol{a})$ is calculated while attempting to achieve the requested relative error bound. The default is 5.
Constraint: **rel_iter** > 0.

out_unit: integer, optional, input
A unit to which the output is written.
If this optional argument is present then the current values of the accuracy and workload variables, whether user specified or default, are written to the given output unit. If not present, then no action is taken. (This argument is more likely to be used for debugging or other special purposes rather than for normal use.)
Constraint: **out_unit** should specify a preconnected unit such as the standard output unit, or a unit to which a file has already been connected. The subroutine does not check such a connection is present, nor for other possible problems such as a file having the "read only" attribute. However, a write error is reported to the standard output unit if the subroutine is unable to write data to the given unit (as detected via a non-zero return value for the IOSTAT variable). Note that if one tries to write to a unit

not connected to a file, some compilers generate a default file and write the data to that file.

r_interval_p(2), r_interval_m(2): real, optional, input

Each of these arguments is an array whose two entries specify the end points (lower, upper; respectively) for the initial intervals of the iteration. The initial breakpoint is then taken as the midpoint of this interval. If present, **r_interval_p** specifies the interval for the "plus" $[I_n^+(\boldsymbol{a})]$ part of the calculation, while **r_interval_m** applies to the "minus" $[I_n^-(\boldsymbol{a})]$ part of the calculation. It is unnecessary to have separate intervals applying to the real and imaginary parts; the same interval **r_interval_p** (or **r_interval_m**) is used for each case.

Defaults: $[0, R_0^{\pm}]$ where $R_0^{\pm}$ are the upper bounds. (Note: in the default case the bounds $R_0^{\pm}$ are taken as the initial breakpoints $R^{\pm}$.)

Constraints:

$$\begin{aligned} 0.0 &\leq \mathbf{r\_interval\_p}(1), \mathbf{r\_interval\_m}(1) \\ \mathbf{r\_interval\_p}(1) &\leq \mathbf{r\_interval\_p}(2) \\ \mathbf{r\_interval\_m}(1) &\leq \mathbf{r\_interval\_m}(2). \end{aligned}$$

Error Control Flags

The value of **ierror** on entry specifies the action the subroutine takes if it detects an error. All error messages are written to the standard output unit.

ierror = 0:

output error message and terminate execution ("hard failure").

ierror = +1:

continue execution, without outputting error messages, returning control back to the calling program ("quiet soft failure"). The error is recorded only by the non-zero return status of **ierror**.

ierror = −1:

output error messages and continue execution, returning control back to the calling program ("noisy soft failure").

ierror = −2:

as −1 but any error messages created by the NAG subroutine D01AKF are also reported ("very noisy soft failure").

Options +1 and −1 suppress the NAG error output, whereas options 0 and −2 enforce it. (Specifically the value of the ifail flag used in calls to D01AKF is +1 and −1, respectively; see the NAG documentation [32].)

Since the subroutine can return useful results even when some type of failure occurs, the recommended choice is to set **ierror** to −1. In more extreme cases, where it may be useful to see which of the twelve component quadratures are causing specific problems, the setting −2 may be more appropriate. The setting +1 may be used for a large batch of repeated calls to the subroutine, provided the return value of **ierror** is always examined.

NOTE: for all non-zero settings of ierror it is essential for the calling program to check the return value of ierror and act accordingly.

Also: if, on entry, **ierror** is not assigned one of the above values then an error message is output and execution is immediately terminated.

Error Return Flags

The value of **ierror** on return indicates the success or otherwise of the calculation. If an error occurs in the subroutine then execution is terminated or a return is made to the calling program, as described above. The output/result arguments will either contain the partial results obtained at that stage of the calculation or, if no such values are available, will be assigned default values (usually zero). The error is indicated via a non-zero return value for the error flag, together with a warning written to the standard output unit. The return values for **ierror** indicate the following scenarios.

ierror $= 0$**:**
normal successful termination of the subroutine.

ierror $= 1$**:**
the maximum number of iterations (**r_iter**) has been reached while refining the breakpoints $R^{\pm}$ without the accuracy requirements being achieved. Try relaxing the accuracy requirements (**err_eps**), increasing the amount of workspace (**lw**,**liw**), or increasing the number of iterations (**r_iter**).

ierror $= 2$**:**
the quadratures along $C_3^{\pm}$ failed. The upper bounds $R_0^{\pm}$ used for the breakpoints $R^{\pm}$ should theoretically prevent this from happening. This flag is included mainly as a safety check. If the user specifies an over-ambitious absolute or relative error tolerance then this error may occur. Likewise, if the absolute tolerance calculated by the routine when attempting to satisfy a relative error bound is exceptionally small then this (and/or other) errors may result. Refer also to the **ierror** $= 5$ case below.

ierror $= 3$**:**
the quadratures along $C_1^{\pm}$ and $C_2'^{\pm}$ failed simultaneously and the adaptive contour algorithm was therefore unable to refine the breakpoint $R^{\pm}$ and continue. This flag is included mainly as a safety check. It should not normally occur for reasonable accuracy requirements and input parameters. If it does, then the best option is to relax the accuracy requirements (**err_eps**) or to increase the amount of workspace (**lw**,**liw**). To succeed, the algorithm essentially has to avoid extreme exponential behaviour along $C_2'^{\pm}$ and therefore must determine a larger breakpoint $R^{\pm}$. In doing so, it must work harder to cope with the increased oscillatory behaviour along $C_1^{\pm}$.

ierror $= 4$**:**

the maximum number of calculations of $C_n(\boldsymbol{a})$ (**rel_iter**) were performed without the relative accuracy requirements being achieved. (However, all quadratures were successful and achieved a certain absolute accuracy, only each time the relative accuracy was not reached.) Try relaxing the accuracy requirements (**err_eps**), increasing the amount of workspace (**lw**,**liw**), or increasing the number of iterations (**rel_iter**). (Note that in the absolute error case this message should not occur. However, the algorithm still checks that the absolute error bound which it calculates is within the requested tolerance and, if not, then this error flag is returned and indicates a problem which should be reported to the authors.)

ierror = 5:
an absolute error tolerance of "zero" is required to achieve the relative error bound that has been requested. The subroutine calculates the required absolute tolerance based on previous approximations to $C_n(\boldsymbol{a})$ and this error will arise, for example, if $C_n(\boldsymbol{a})$ has a value of "zero". Rather than trying to approximate $C_n(\boldsymbol{a})$ for such an unreasonable absolute error tolerance and producing numerous errors in the process, the subroutine returns this error flag instead. More precisely, the subroutine does not check for an absolute tolerance of "zero" (by which we mean one so small that it is regarded as zero by the computer). Rather it checks for an exceptionally small absolute error tolerance defined by the principle that the quadrature subroutines should not be expected to cope given such a tolerance. This bound is specified in the program code by a constant (**tol_inf**) whose default value is 10^{-20}.

The following error flags indicate minor errors such as invalid argument values.

ierror = 10:
on entry, **deriv** < 0 or **deriv** $> n - 2$

ierror = 11:
on entry, **err_eps** < 0.0 or **err_eps** $<$ "tiny", the smallest possible real. (That is, **err_eps** is negative or too small respectively.)

ierror = 12:
on entry, **liw** < 1 or **lw** < 4 **liw**

ierror = 13:
on entry, **r_iter** ≤ 0 or **rel_iter** ≤ 0

ierror = 14:
on entry, the shape of array **r_values** is incorrect and/or not consistent with the value of **r_iter** (whether user specified or default). The array **r_values** should have dimensions (1:2,1:2,0:**r_iter**).

ierror = 15:
on entry, **r_interval_p**(1) < 0.0 or **r_interval_p**(1) $>$ **r_interval_p**(2) and similarly for **r_interval_m**.

The following error flags indicate that the subroutine was unable to allocate the space required by dynamic arrays. Under normal circumstances the sizes of the arrays in question are small and such problems should not occur. These flags are included mainly as a safety check.

ierror = 20**:**
could not allocate the array **a_g** used to store global copies of the parameters **a**(i).

ierror = 21**:**
the driver subroutine **cuspint** could not allocate the array **r_values** needed in the call to **cuspint_e**. This is actually an error returned by the driver subroutine and not by the subroutine **cuspint_e** itself; it is included here for convenience.

5 Test Runs and Examples

This section provides three examples of the use of the **cuspint** module. The three examples are (1) a driver program for the Pearcey integral and its two partial derivatives which outputs a table of values for these three integrals. (2) gray shaded perspective views of the modulus of the swallowtail integral calculated using **cuspint**, and (3) an analysis of more difficult computations in which the adaptive contour algorithm is employed and examples where changes to the default are essential for a successful calculation.

5.1 Pearcey Integral and its Derivatives

Figure 2 shows a driver program for the Pearcey integral

$$P(x,y) = \int_{-\infty}^{\infty} \exp[\mathrm{i}(u^4 + xu^2 + yu)]\,\mathrm{d}u \tag{16}$$

and its two first order partial derivatives

$$\frac{\partial P(x,y)}{\partial x} = \mathrm{i}\int_{-\infty}^{\infty} u^2 \exp[\mathrm{i}(u^4 + xu^2 + yu)]\,\mathrm{d}u \tag{17}$$

and

$$\frac{\partial P(x,y)}{\partial y} = \mathrm{i}\int_{-\infty}^{\infty} u \exp[\mathrm{i}(u^4 + xu^2 + yu)]\,\mathrm{d}u. \tag{18}$$

```
! PROGRAM testrun
!
! Test run program for the cuspoid integral module. Tabulates values
! of Pearcey's integral P(x,y) and its two partial derivatives.
!
! Form of the integrand: exp(I*(u^4+x*u^2+y*u))
! Grid: x = -8.0 (2.0) 8.0, y = 0.0 (2.0) 8.0
!
! Data is written to the file testrun.dat. Note: this file is created
! by the program - it will return an error if the file already exists.

program testrun
   use cuspoid_integral
   implicit none

   ! unit used to write output
   integer, parameter :: out_unit=8

   real(kind=Rkind) :: abserr
   real(kind=Rkind), dimension(2) :: a,p,dpx,dpy
   integer :: ix,iy,deriv,ierror

   open(unit=out_unit,file="testrun.dat",status="new",action="write")

   write(out_unit,'("Pearcey''s integral P(x,y) and its derivatives")')
   write(out_unit,'(1X)')
   write(out_unit,'(T5,"x",T12,"y",T19,"Re P(x,y)",T30,"Im P(x,y)", &
      & T41,"Re dP/dx",T52,"Im dP/dx",T63,"Re dP/dy",T74,"Im dP/dy")')

   ! non-integer do-loop variables obsolescent in Fortran 90
   ! use integers and cast to reals before calling cuspint routine
   do iy = 0,8,2
      do ix = -8,8,2

         a(1) = real(iy,Rkind)
         a(2) = real(ix,Rkind)

         deriv = 0
         ierror = -1
         call cuspint(a,deriv,p,abserr,ierror)
         call error_message()

         deriv = 2
         ierror = -1
         call cuspint(a,deriv,dpx,abserr,ierror)
         call error_message()

         deriv = 1
         ierror = -1
         call cuspint(a,deriv,dpy,abserr,ierror)
         call error_message()

         write(out_unit,'(2F7.1,6F11.5)') a(2),a(1),p,dpx,dpy

      end do
   end do

contains
```

Fig. 2. Driver program to produce a table of values for the real and imaginary parts of $P(x,y)$, $\partial P(x,y)/\partial x$ and $\partial P(x,y)/\partial y$ on the grid $x = -8.0$ (2.0) 8.0 and $y = 0.0$ (2.0) 8.0. This program is contained in the file: **testrun.f90**. (Continued over …)

```
! INTERNAL SUBROUTINE error_message
!
! If cuspint returns an error this routine logs a message in the output
! file accordingly.
!
! Note that the value of ierror used in the call to cuspint will also
! cause an error message to be sent to the standard output unit.
!
subroutine error_message()

   if (ierror /= 0) then
      write(out_unit,*) "*** error returned"
      write(out_unit,*) "*** x,y,deriv,ierror: ",a(2),a(1),deriv,ierror
   end if

end subroutine error_message

end program testrun
```

Fig. 2 (continued).

The driver program produces the table of values shown in Table 1 for the real and imaginary parts of $P(x, y)$, $\partial P(x, y)/\partial x$ and $\partial P(x, y)/\partial y$ on the grid $x = -8.0\ (2.0)\ 8.0$ and $y = 0.0\ (2.0)\ 8.0$. Note: the driver program is contained in the file **testrun.f90** in the package; it writes data to the file **testrun.dat**. Observe that it is only necessary to consider $y \geq 0$ because of the relations

$$P(x, -y) = P(x, y) \tag{19}$$

$$\frac{\partial P(x, -y)}{\partial x} = \frac{\partial P(x, y)}{\partial x} \tag{20}$$

$$\frac{\partial P(x, -y)}{\partial y} = -\frac{\partial P(x, y)}{\partial y}. \tag{21}$$

The results in Table 1 can be compared directly with those in Table 1 of reference [35], which have been used to test other numerical procedures for the computation of $P(x, y)$ [3,39,40]. On making the comparison, it is found that every digit agrees, the only exception being a typographical error in [35] for the sign of the entry for Re $\partial P(x, y)/\partial x$ at $x = -8.0$, $y = 8.0$ where -1.69620 should read $+1.69620$. (This error has been corrected in Table 2 of [40].)

As a further check, we have also verified the results in Table 1 using Maple 5.3 on a Sun Sparc Ultra 2 Workstation. The computations took about 4 hours whereas the corresponding calculations using **cuspint** on the same computer took a few seconds.

Finally we note that the contour integral method of [35] has been used to compute $P(x, y)$ by Fang et al. [41] who study the focussing of surface acoustic waves, by Berry [42] and by Berry and Klein [43] who analyse optical caustics, and by Hanyga and Seredyńska [44] who extend the geometrical theory of

```
Pearcey's integral P(x,y) and its derivatives

    x      y      Re P(x,y)  Im P(x,y)  Re dP/dx   Im dP/dx   Re dP/dy   Im dP/dy
   -8.0   0.0   -0.33744   -0.87636    1.69277   -3.15815    0.00000    0.00000
   -6.0   0.0    0.15928   -1.48342    2.90365   -1.14939    0.00000    0.00000
   -4.0   0.0   -0.64756   -0.60962   -0.16343   -2.58817    0.00000    0.00000
   -2.0   0.0    2.38566   -1.08551    0.54189    1.52244    0.00000    0.00000
    0.0   0.0    1.67481    0.69373   -0.56607    0.23447    0.00000    0.00000
    2.0   0.0    0.92403    0.72901   -0.21594   -0.06998    0.00000    0.00000
    4.0   0.0    0.64698    0.59370   -0.08696   -0.05763    0.00000    0.00000
    6.0   0.0    0.52085    0.50005   -0.04595   -0.03760    0.00000    0.00000
    8.0   0.0    0.44792    0.43762   -0.02909   -0.02592    0.00000    0.00000
   -8.0   2.0    1.00422   -0.11480   -0.88749    2.11347   -1.17531   -0.48474
   -6.0   2.0    0.96527    0.46413   -2.55501    1.26598   -0.19755   -0.39687
   -4.0   2.0    1.96341   -0.73419    0.59863    2.27823    0.71924    0.12763
   -2.0   2.0    0.35455   -0.05184    0.11605   -0.78569   -1.18612    0.91510
    0.0   2.0    1.12475   -0.17608    0.13922    0.34275   -0.59981   -0.62528
    2.0   2.0    0.99372    0.31273   -0.14711    0.11925    0.00417   -0.39559
    4.0   2.0    0.74010    0.41332   -0.09909    0.00836    0.07104   -0.18521
    6.0   2.0    0.58773    0.40353   -0.05768   -0.01200    0.05842   -0.10067
    8.0   2.0    0.49582    0.37668   -0.03657   -0.01373    0.04389   -0.06352
   -8.0   4.0    0.75372   -0.23933   -0.15757    0.44099    1.77246    0.58428
   -6.0   4.0    0.29478   -0.84373    1.95637   -1.63566    0.79228    1.00121
   -4.0   4.0    0.14360    0.90244   -0.77799   -1.37935   -1.06948    1.04356
   -2.0   4.0    0.08086    0.89242   -1.02267    0.01130    0.61242   -0.41420
    0.0   4.0   -0.38592   -0.54514    0.57492   -0.50984   -0.61187    0.41959
    2.0   4.0    0.59648   -0.56516    0.25310    0.24219   -0.44840   -0.34574
    4.0   4.0    0.76660   -0.13266   -0.01609    0.15887   -0.09643   -0.33414
    6.0   4.0    0.68391    0.08129   -0.05079    0.06657    0.01127   -0.21989
    8.0   4.0    0.58882    0.16933   -0.04264    0.02734    0.03539   -0.14613
   -8.0   6.0   -0.12839    0.34848   -0.12600   -2.76756   -1.12322    0.24977
   -6.0   6.0    1.17888    1.08442   -0.72855    2.18822   -1.58555   -0.12079
   -4.0   6.0    0.04838    0.24046    0.31562    0.95464   -0.04381   -1.32236
   -2.0   6.0    0.02399   -0.53796    1.19911    0.15601   -0.73138   -0.16795
    0.0   6.0   -0.23537    0.59203   -0.77556   -0.26813    0.68961    0.22520
    2.0   6.0   -0.47683   -0.50921    0.35819   -0.36844   -0.41183    0.44540
    4.0   6.0    0.22551   -0.66816    0.25158    0.10091   -0.42738   -0.10886
    6.0   6.0    0.51590   -0.40573    0.06433    0.12934   -0.20150   -0.22859
    8.0   6.0    0.56595   -0.19254   -0.00103    0.08381   -0.07757   -0.20209
   -8.0   8.0    1.06930    0.22585    1.69620    3.11697   -0.52459    0.41649
   -6.0   8.0   -1.10157    0.58229   -1.16047   -1.60247    0.90985   -0.72015
   -4.0   8.0   -0.49013    0.02199   -0.41976   -1.19611    0.47430    0.79769
   -2.0   8.0   -0.18003    0.46915   -1.12562   -0.36724    0.75145    0.26501
    0.0   8.0    0.51018   -0.26097    0.44181    0.79903   -0.34929   -0.63475
    2.0   8.0   -0.30892    0.54515   -0.55335   -0.29050    0.56095    0.28408
    4.0   8.0   -0.56703   -0.30814    0.18333   -0.33249   -0.21410    0.44940
    6.0   8.0   -0.09657   -0.61455    0.21947   -0.01577   -0.36265    0.07405
    8.0   8.0    0.22986   -0.53241    0.11029    0.07181   -0.25522   -0.09996
```

Table 1
Real and imaginary parts of $P(x,y)$, $\partial P(x,y)/\partial x$ and $\partial P(x,y)/\partial y$ on the grid $x = -8.0$ (2.0) 8.0 and $y = 0.0$ (2.0) 8.0 as computed by the driver program in Figure 2. This data is written to the file: **testrun.dat**.

diffraction to take into account caustics. The asymptotics of $P(x,y)$ have been investigated by Paris [45–47] and by Kaminski [48]. These authors have also made a detailed study [49] of the zeroes of $P(x,y)$ computed by the differential equation method [18,22–24] that was mentioned in Section 1.

*** Swallowtail plot for $x = 4.0$ here. ***

Fig. 3. Gray shaded perspective plot of $|S(x, y, z)|$ for $x = 4.0$. The numbers on the caustic section are the number of real stationary phase points in different regions of (x, y, z) space.

5.2 Swallowtail Integral

We have also used **cuspint** to compute values for the swallowtail integral defined by

$$S(x, y, z) = \int_{-\infty}^{\infty} \exp[\mathrm{i}(u^5 + xu^3 + yu^2 + zu)]\, \mathrm{d}u. \tag{22}$$

Figures 3–5 show gray shaded perspective views of $|S(x, y, z)|$ for the fixed values of $x = 4.0$, 0.0 and -6.0 respectively. The grids used in the plots are -20.0 (0.3) 29.8 for z and -20.0 (0.3) 19.9 for y. Thus each plot requires 22378 numerical evaluations of $S(x, y, z)$. The plots are symmetric about the line $y = 0$ because of the relation

$$S(x, -y, z) = S^*(x, y, z). \tag{23}$$

Also shown in Figures 3–5 are sections of the caustic surface associated with the swallowtail catastrophe for $x = 4.0$, 0.0 and -6.0, respectively. The caustic surface is obtained by eliminating real values of u from the equations

$$5u^4 + 3xu^2 + 2yu + z = 0 \tag{24}$$
$$20u^3 + 6xu + 2y = 0. \tag{25}$$

The caustic is symmetric about $y = 0$, like $|S(x, y, z)|$. The numbers on the caustic sections are the number of real solutions of equations (24) and (25) [i.e. the number of real stationary phase points for the exponent in the integrand of $S(x, y, z)$] in different regions of (x, y, z) space. The caustic sections act as "skeletons" upon which is built the "wave flesh" associated with $S(x, y, z)$. The asymptotics of $S(x, y, z)$ near the cusp of the caustic have been studied by Kaminski [50].

Our plots for $|S(x, y, z)|$ agree with those reported earlier in references [21] and [23] (which however used a less dense grid), where a detailed discussion of the structure in the plots can be found.

*** Swallowtail plot for $x = 0.0$ here. ***

Fig. 4. Gray shaded perspective plot of $|S(x, y, z)|$ for $x = 0.0$. The numbers on the caustic section are the number of real stationary phase points in different regions of (x, y, z) space.

*** Swallowtail plot for $x = -6.0$ here. ***

Fig. 5. Gray shaded perspective plot of $|S(x, y, z)|$ for $x = -6.0$. The numbers on the caustic section are the number of real stationary phase points in different regions of (x, y, z) space.

5.3 More Difficult Calculations

In this section, we consider calculations which are more challenging than those reported above. Our purpose is to demonstrate the way the algorithm adapts the contours and to give examples where changes to the defaults are essential for a successful calculation. The discussion is illustrative rather than comprehensive.

We consider in the four subsections that follow $P(x, x)$ and $S(x, x, x)$ for increasing positive x and for decreasing negative x. The calculations were all performed using the NAG version of **cuspint** on a Sun Sparc Ultra 2 Workstation, together with the NAG FORTRAN 90 compiler. We used the defaults **err_eps** $= 10^{-5}$, **err_abs** = **true**, **lw** $= 800$, **liw** $= 200$, **r_iter** $= 5$, and **rel_iter** $= 5$ except where indicated. All our results are summarised in Table 2. Note that each example is for a *specific* component of the calculation. For example, for $P(x, x)$ with x large and positive, it is for the Re $I_4^-(x, x)$ component. Similar considerations may (or may not) apply to the other components.

The results presented in this section were all obtained using the driver program **cuspint_dr2** supplied with the package.

5.3.1 $P(x, x)$ for large positive x

Rows a-d in Table 2 report our results for Re $I_4^-(x, x)$. First we note that the default set of contours (with a single breakpoint) is sufficient for $x \leq 422.0$, but for $x = 423.0$ the adaptive contour algorithm is required. As x increases, rows a-c show that the Cauchy bound calculated by the algorithm [see Section 3.3 (1)] becomes less sharp and the iteration produces breakpoints R^- which decrease monotonically to a value near zero. The subroutine returns an error (**ierror** = 1: maximum number of iterations reached while trying to obtain a suitable breakpoint) for some x between 16900.0 and 17000.0, see row c. Of course, one could calculate $P(x, x)$ for $x = 17000.0$ by increasing the number of iterations (**r_iter**). However, upon examining the breakpoints

Table 2
Results from **cuspint** demonstrating the adaptive contour algorithm and examples where changing the defaults leads to a successful calculation.

row	x	default	**ierror**	iterates of R^+ or R^-				
Re $I_4^-(x,x)$ component of $P(x,x)$								
a	423.0	Yes	0	5.46	2.73	—	—	—
b	1063.0	Yes	0	7.42	3.71	1.85	—	—
c	17000.0	Yes	1	18.67	9.33	4.67	2.33	1.17
d	17000.0	No	0	0.58	—	—	—	—
Re $I_4^+(x,x)$ component of $P(x,x)$								
e	-101.0	Yes	0	11.95	5.98	8.96	—	—
f	-176.0	Yes	0	15.78	7.89	11.83	9.86	—
g	-202.0	Yes	3	16.90	8.45	—	—	—
h	-202.0	No	0	16.90	8.45	12.68	10.56	—
i	-202.0	No	0	12.00	—	—	—	—
Re $I_5^-(x,x,x)$ component of $S(x,x,x)$								
j	148.0	Yes	0	4.43	2.22	—	—	—
k	9700.0	Yes	1	17.86	8.93	4.47	2.23	1.12
l	9700.0	No	0	0.56	—	—	—	—
Re $I_5^+(x,x,x)$ component of $S(x,x,x)$								
m	-22.0	Yes	0	7.31	3.65	—	—	—
n	-70.0	Yes	0	13.03	6.52	9.78	8.15	7.33
o	-71.0	Yes	1	13.13	6.56	9.85	8.20	7.38
p	-71.0	No	0	7.00	—	—	—	—

in row c, a more sensible approach is to override the default initial interval for R^- by specifying **r_interval_m** $= (0.0, 1.17)$, and row d shows that the calculation can now be done. Note that adapting the contours is essential for the success of this calculation. For $x = 17000.0$ the integrand along $C_2'^-$ exhibits extremely violent exponential behaviour which is concentrated very close to the origin. Thus, if one blindly chooses **r_interval_m** $= (0.0, 0.0)$ then this choice of initial breakpoint, $R^- = 0.0$, positions the C_3^- contour right in the middle of the extreme exponential region and the program aborts with a floating point overflow exception.

5.3.2 $P(x, x)$ for large negative x

Rows e - i of Table 2 show results obtained during the calculation of Re $I_4^+(x, x)$. Observe that as x becomes more negative, the iteration produces a ‘spiralling’ sequence of breakpoints R^+. For $x = -202.0$ and the second iterate $R^+ = 8.45$, row g of Table 2 shows that the subroutine returns an error (**ierror** $= 3$: quadrature failed along the C_1^+ and $C_2'^+$ contours; unable to refine the breakpoint). For this example, the subroutine returns a ‘result’ of the order of 10^{251} (with an absolute error estimate of 10^{250}), which indicates just how badly behaved is the integrand for this choice of R^+. In order for the calculation to succeed, we have to increase the workspace parameters, so that the NAG subroutine can perform the quadrature along C_1^+. This is achieved by setting **lw** $= 1000$, **liw** $= 250$ (all other optional arguments are set to their default values); see row h. In addition, the previous (failed) calculation in row g suggests it makes sense to override the default initial interval for R^+ by trying, say, **r_interval_p** $= (8.0, 16.0)$. This leads to an immediate success with $R^+ = 12.00$; see row i. [Note for $x = -202.0$, the default value for R^- of 11.95 is sufficient. Thus, if the default interval for R^- is overridden, it sufficient to specify, say, **r_interval_m** $= (11.0, 12.0)$ in **cuspint_dr2**.]

We also remark that, as x becomes more negative, the need for user intervention increases. The main problem to avoid is regions where the integrand becomes highly exponential, as these can cause the program to abort with an arithmetic overflow, before the algorithm can modify its choice of $R^\pm$. Thus, based on the results of previous calculations, we can help the algorithm perform a successful computation by specifying initial intervals for the $R^\pm$.

5.3.3 $S(x, x, x)$ for large positive x

Rows j-l of Table 2 display our results for the Re $I_5^-(x, x, x)$ component of $S(x, x, x)$ for large positive x. We find a similar situation to the Pearcey case in rows a-c, with the sequence of breakpoints R^- decreasing monotonically to a value near zero. An error is finally returned (**ierror** $= 1$) for some x between 9600.0 and 9700.0. However, row l shows that specifying **r_interval_m** $= (0.0, 1.12)$ leads to a successful computation for $x = 9700.0$.

5.3.4 $S(x, x, x)$ for large negative x

Our results for the Re $I_5^+(x, x, x)$ component of $S(x, x, x)$ are reported in rows m-p of Table 2. As x becomes more negative, the iteration produces a ‘spiralling’ sequence of breakpoints R^+, similar to the Pearcey case in rows e and f. At $x = -71.0$ the maximum number of iterations is reached resulting in an error (**ierror** $= 1$). However, row p shows that setting **r_interval_p** $= (6.0, 8.0)$ enables the computation to be done successfully.

Acknowledgements

The support of this research by the UK Engineering and Physical Sciences Research Council is gratefully acknowledged.

References

[1] J.N.L. Connor, Practical methods for the uniform asymptotic evaluation of oscillating integrals with several coalescing saddle points, in: *Asymptotic and Computational Analysis*, Conference in Honor of Frank W.J. Olver's 65th Birthday, ed. R. Wong (Dekker, New York, 1990) 137–173.

[2] J.N.L. Connor, P.R. Curtis, R.A.W. Young, Uniform asymptotics of oscillating integrals: applications in chemical physics, in *Wave Asymptotics*, The proceedings of the meeting to mark the retirement of Professor Fritz Ursell from the Beyer Chair of Applied Mathematics in the University of Manchester, edited by P.A. Martin and G.R. Wickham (Cambridge University Press, Cambridge, 1992) 24–42.

[3] J.J. Stamnes, *Waves in Focal Regions: Propagation, Diffraction and Focusing of Light, Sound and Water Waves*, (Hilger, Bristol, 1986).

[4] V.M. Babič, V.S. Buldyrev, *Short Wavelength Diffraction Theory: Asymptotic Methods*, translated from the Russian by E.F. Kuester, (Springer, Berlin, 1991).

[5] L.M. Brekhovskikh, O.A. Godin, *Acoustics of Layered Media II Point Sources and Bounded Beams*, second updated and enlarged edition, (Springer, Berlin, 1999).

[6] P.L. Marston, Geometrical and catastrophe optics methods in scattering, *Physical Acoustics*, **21** (1992) 1–234.

[7] V.A. Borovikov, B. Ye Kinber, *Geometrical Theory of Diffraction*, (Institution of Electrical Engineers, London, 1994).

[8] Yu. A. Kravtsov, Yu. I. Orlov., *Caustics, Catastrophes and Wave Fields*, 2nd ed., (Springer, Berlin, 1999).

[9] J.N.L. Connor, Semiclassical theory of molecular collisions: three nearly coincident classical trajectories, *Mol. Phys.*, **26** (1973) 1217–1231.

[10] J.N.L. Connor, Semiclassical theory of molecular collisions. Many nearly coincident classical trajectories, *Mol. Phys.*, **27** (1974) 853–866.

[11] J.N.L. Connor, Catastrophes and molecular collisions, *Mol. Phys.*, **31** (1976) 33–55.

[12] R. Wong, *Asymptotic Approximations of Integrals*, (Academic, Boston, 1989).

[13] V.A. Borovikov, *Uniform Stationary Phase Method*, (Institution of Electrical Engineers, London, 1994).

[14] N.M. Temme, Uniform asymptotic expansions of integrals: a selection of problems, *J. Comput. Appl. Math.*, **65** (1995) 395–417.

[15] G.H. Hardy, On certain definite integrals considered by Airy and Stokes, *Quart. J.*, **41** (1910) 226–240, [reprinted in: *Collected Papers of G.H. Hardy*, Vol. IV, edited by a Committee appointed by The London Mathematical Society, (Clarendon, Oxford, 1969) pp. 460–474].

[16] V.I. Arnol'd, Normal'nye formy funktsiy v okrestnosti vyrozhdennykh kriticheskikh tochek (Normal forms of functions in neighborhoods of degenerate critical points), *Usp. Mat. Nauk.*, **29** (2) (1974), 11–49. [English translation: *Russ. Math. Surv.*, **29** (2) (1974), 10–50. Also in: V.I. Arnold, *Singularity Theory: Selected Papers*, (Cambridge University Press, Cambridge, 1981), pp. 91-131.]

[17] T. Poston, I. Stewart, *Catastrophe Theory and its Applications*, (Pitman, London, 1978).

[18] J.N.L. Connor, D. Farrelly, Theory of cusped rainbows in elastic scattering: Uniform semiclassical calculations using Pearcey's integral, *J. Chem. Phys.*, **75** (1981) 2831–2846.

[19] F. Wolf, H.J. Korsch, Fold and cusp catastrophe structures in rotational rainbow scattering, *J. Chem. Phys.*, **81** (1984) 3127–3136.

[20] J.N.L. Connor, P.R. Curtis, C.J. Edge, A. Laganà, The uniform asymptotic swallowtail approximation: Application to the collinear $H + F_2$ chemical reaction, *J. Chem. Phys.*, **80** (1984) 1362–1363.

[21] J.N.L. Connor, P.R. Curtis, D. Farrelly, The uniform asymptotic swallowtail approximation: practical methods for oscillating integrals with four coalescing saddle points, *J. Phys. A.*, **17** (1984) 283–310.

[22] J.N.L. Connor, D. Farrelly, Molecular collisions and cusp catastrophes: Three methods for the calculation of Pearcey's integral and its derivatives, *Chem. Phys. Letters*, **81** (1981) 306–310.

[23] J.N.L. Connor, P.R. Curtis, D. Farrelly, A differential equation method for the numerical evaluation of the Airy, Pearcey and swallowtail canonical integrals and their derivatives, *Mol. Phys.*, **48** (1983) 1305–1330.

[24] J.N.L. Connor, P.R. Curtis, Differential equations for the cuspoid canonical integrals, *J. Math. Phys.*, **25** (1984) 2895–2902.

[25] P.J. Davis, P. Rabinowitz, *Methods of Numerical Integration*, 2nd edition, (Academic, San Diego, 1984).

[26] G. Evans, *Practical Numerical Integration*, (Wiley, New York, 1993).

[27] A.R. Krommer, C.W. Ueberhuber, *Computational Integration*, (SIAM, Philadelphia, 1998).

[28] K.B. Winterbon, Comment on "Comparison of Some Methods for Evaluating Infinite Range Oscillatory Integrals", *J. Comp. Phys.*, **26** (1978) 447–448.

[29] R. Lugannani, S.O. Rice, Numerical evaluation of a class of integrals by integrating along a string of saddle points, *J. Comp. Phys.*, **39** (1981) 473–480.

[30] G.A. Evans, K.C. Chung, The evaluation of infinite range oscillatory integrals using optimal contours in the complex plane, *Int. J. Comput. Math.*, **66** (1998) 39–52.

[31] J. Rasch, C.T. Whelan, Partial wave integrals, in *High Performance Computing*, edited by R.J. Allan, M.F. Guest, A.D. Simpson, D.S. Henty and D.A. Nicole, (Kluwer Academic, New York, 1999) pp. 301–305.

[32] Numerical Algorithms Group, FORTRAN *Library Manual*, Mark 16, Vol. 1, subroutine D01AKF, (NAG, Oxford, 1993).

[33] R. Piessens, E. de Doncker-Kapenga, C.W. Überhuber, D.K. Kahaner, *QUADPACK, A Subroutine Package for Automatic Integration*, (Springer, Berlin, 1983).

[34] M.A. Malcolm, R.B. Simpson, Local versus global strategies for adaptive quadrature, *ACM Trans. Math. Software*, **1** (1975) 129–146.

[35] J.N.L. Connor, P.R. Curtis, A method for the numerical evaluation of the oscillatory integrals associated with the cuspoid catastrophes: application to Pearcey's integral and its derivatives, *J. Phys. A*, **15** (1982) 1179–1190.

[36] R. Benedetti, J.-J. Risler, *Real Algebraic and Semi-Algebraic Sets*, (Hermann, Paris, 1990).

[37] D.S. Lukin, E.B. Ipatov, E.A. Palkin, Algoritm chislennogo rascheta spetsial'nykh funktsiy tipa bystro-ostsilliruyushchikh integralov (An algorithm for the numerical evaluation of special functions of the rapidly oscillating integral type), in: *Voprosy Difraktsii Elektromagnitnykh Voln (Questions on the Diffraction of Electromagnetic Waves)*, (MFTI, Moscow, 1982) 21–35.

[38] E.B. Ipatov, D.S. Lukin, E.A. Palkin, Chislennye metody pascheta spetsial'nykh funktsiy volnovykh katastrof (Numerical methods of computing special functions of wave catastrophes), *Zh. Vychisl. Mat. Mat. Fiz.*, **25** (1985) 224–236. [English translation: *USSR Comput. Math. Math. Phys.*, **25** (1985) 144–153.]

[39] A. Druzhinin, H. Pedersen, M. Campillo, W.H. Kim, Elastic Kirchhoff-Helmholtz synthetic seismograms, *Pure Applied Geophys.*, **151** (1998) 17–45.

[40] J.J. Stamnes, B. Spjelkavic, Evaluation of the field near a cusp of a caustic, *Optica Acta*, **30** (1983) 1331–1358.

[41] S.R. Fang, S.Y. Zhang, Z.F. Lu, SAW Focusing by Circular-Arc Interdigital Transducers on YZ–LiNbO$_3$, *IEEE Trans. Ultrasonics, Ferroelectrics and Frequency Control*, **36** (1989) 178–184.

[42] M. Berry, Rays, wavefronts and phase: a picture book of cusps, in *Huygens' Principle*, 1690–1990, *Theory and Applications*, Proceedings of an International Symposium, The Hague/Scheveningen, November 19–22, 1990, edited by H. Blok, H.A. Ferweda and H.K. Kuiken, (North Holland, Amsterdam, 1992) pp. 97–111.

[43] M.V. Berry, S. Klein, Colored diffraction catastrophes, *Proc. Natl. Acad. Sci. USA*, **93** (1996) 2614–2619.

[44] A. Hanyga and M. Seredyńska, Diffraction of pulses in the vicinity of simple caustics and caustic cusps, *Wave Motion*, **14** (1991) 101–121.

[45] R.B. Paris, The Aysmptotics of Pearcey's integral for complex variables, in *Asymptotic and Computational Analysis*, Conference in Honor of Frank W.J. Olver's 65th Birthday, edited by R. Wong, (Dekker, New York, 1990) pp. 653–667.

[46] R.B. Paris, The asymptotic behaviour of Pearcey's integral for complex variables, *Proc. Roy. Soc. London A*, **432** (1991) 391–426.

[47] R.B. Paris, A generalization of Pearcey's integral, *SIAM J. Math. Anal.*, **25** (1994) 630–645.

[48] D. Kaminski, Asymptotic evaluation of the Pearcey integral near the caustic, *SIAM J. Math. Anal.*, **20** (1989) 987–1005.

[49] D. Kaminski, R.B. Paris, On the zeroes of the Pearcey integral, *J. Comput. Appl. Math.*, **107** (1999) 31–52.

[50] D. Kaminski, Asymptotics of the swallowtail integral near the cusp of the caustic, *SIAM J. Math. Anal.*, **23** (1994) 262–285.