SCRUMPI
Sound **C**alculation for **R**otation of **U**nsteady-source **M**ultibladed **P**ropellers with **I**ncidence
Version 0.97

DRAFT

Michael Carley

NO WARRANTY

This code was written for research purposes. It is not intended as commercial software and it comes with absolutely no warranty, including, but not limited to, the implied warranty of fitness for a given purpose.

In particular neither the author nor Trinity College, Dublin nor the Department of Mechanical Engineering, TCD take any responsibility for this code or for the results of its use.

Furthermore, this code is not, under any circumstances, licenced for any military or 'defence' use.

Contents

List of Figures

List of Tables

Chapter 1

SCRUMPI

SCRUMPI (**S**ound **C**alculation for **R**otation of **U**nsteady-source **M**ultibladed **P**ropellers with **I**ncidence) is an implementation of an extended version of Farassat's formulation 1A, [1], for predicting the noise from subsonic propellers[1]. At present it handles monopole (thickness) and dipole (loading) noise generated by both steady and fluctating on-blade pressures and velocities. In addition, it is valid for arbitrary geometry and inflow orientation.

I also hope to implement an aerodynamic formulation, similar to Farassat's [2], which is a development of the acoustic formulae to allow calculation of the linear aerodynamics of a propeller. But that's another few months work if I ever get around to it.

1.1 Requirements

The required input data are

1. operating conditions.
2. blade surface geometry.
3. blade surface pressure.

The surface pressure data are specified in the frequency domain and can have arbitrary spectra so that turbulent source noise prediction is possible, given suitable aerodynamic input.

At present the following operating condition data can be specified with three degrees of freedom in Cartesian geometry.

[1] It's also as near as makes no difference the name of a very nice cider, but that's neither here nor there.

1. Inflow velocity.

2. Hub (propeller) position.

3. Hub translational velocity.

4. Orientation of axis of rotation.

5. Observer position.

The current limitations are that the flight path must be one of constant velocity and, more importantly, the propeller must operate acoustically subsonically (i.e. no part of a blade may approach the observer at a velocity greater than the speed of sound). No check is made in the code for this and it will crash (or worse, give you very bad results) if a blade does operate supersonically. It could be some time before I do anything about this.

1.2 Installation

The program is available as C source code. In principle, it is simply a matter of copying the files to an appropriate directory, configuring the makefile and typing `make` which will build the SCRUMPI executable. To date the code has been installed and run succesfully on the following systems.

- Hewlett-Packard HP-UX 8 and 9.
- Sun OS 4.1.3.
- PC running Linux.
- MIPS 4.51[2].

A sample input file called `test.in` is available and typing `make test` after you build SCRUMPI will run a calculation for that file and put the result in `test.out`. The reference result is in a file called `test` so that you can check the program is running as it should.

[2]MIPS cc does not support the %g format specifier in scanf so the code needs a little hacking to compile, see appendix C

1.3 Use

Once you have the input data set up (see appendix B) just run SCRUMPI from the command line. The command takes the form

```
scrumpi <options> <input file>
```

where the available options are shown in table 1.1.

`-a`	write ASCII retarded time data	`-b`	write binary retarded time data (**default**)
`-e#`	end time (**1 revolution**)	`-h`	print help message
`-o*`	output file name (**standard output**)	`-q`	run quietly (**write messages**)
`-r#`	retarded shape steps (§3.4.2)	`-s#`	start time (**0s**)
`-t#`	sample rate for loading record (§B.3.2)	`-A#`	angle of attack (**0**°)
`-B#`	number of blades (**as per input file**)	`-E#`	end time in fractions of a rev
`-H#`	maximum number of loading harmonics to use	`-L#`	number of points in loading record (§B.3.2)
`-N#`	number of points in predicted signal (**128**)	`-P`	use point force approximation (§1.3.2)
`-Q`	don't write time record to file	`-R`	name of retarded shape file (§3.4.2)
`-S#`	start time in fractions of a rev	`-T#`	initial blade azimuth
`-V`	read velocities from file (§B.4)	`-W`	print no-warranty message
`-X#`	observer x coordinate (**10m**)	`-Y#`	observer y coordinate (**0m**)
`-Z#`	observer z coordinate (**0m**)		

Table 1.1: Command line options: options followed by a '#' take a numerical argument, those followed by an asterisk require a text string. **Bold** items are default settings

Options followed by a * take a string argument, those followed by # take a number. In general, operating conditions are set in the input file as they all depend on the "real world" and are not logically user-selected while things like observer position should be easily changed and so are set on the command line. The exception is the `-B#` option which lets you specify a number of blades. This is so that you can decide to take a "single-blade" time record rather than the default superposed, multiblade one.

1.3.1 Default settings

Default settings for the command line options are given in table 1.1. If no input file is specified then SCRUMPI reads from standard input so a perfectly acceptable way of running the code is

```
scrumpi <options> < Input.file > Output.file
```

All these defaults can be set at compile time.

1.3.2 Point force mode

While SCRUMPI is intended for full-surface calculations of propeller noise, it can also be used in a 'point force' mode where the blade is approximated for acoustical purposes by a rotating point source, as in a Gutin [3] approximation. In this case the mesh will be 1×1 points and should be specified as such in the input file. As there is no blade geometry to use in working out the surface normal you also need to specify the direction of the force in the input file. This goes between the geometry specification (i.e. the position of the point force) and the loading data. It is also possible to specify a velocity source (i.e. a monopole) strength in point source mode using the `-V` option.

1.4 Support

There isn't any. That's not quite true—if you have a problem mail me and I will do what I can to help but there is no support contract and there is no guarantee of early assistance (if any). I'm

Michael Carley
Dept. of Mechanical Engineering
Trinity College,
Dublin 2,
IRELAND.

carleym@tcd.ie

Chapter 2

Theory

The fundamental theory of noise generation by solid bodies is the Ffowcs Williams-Hawkings equation, [4] which is an exact formulation for the sound generated by a moving solid body in a flow.

$$
\begin{aligned}
4\pi c^2 \rho'(\mathbf{x},t) = & \frac{\partial^2}{\partial x_i \partial x_j} \int_V \left[\frac{T_{ij}}{r|1-M_r|} \right] d^3\mathbf{y} \\
& - \frac{\partial}{\partial x_j} \int_S n_i \left[\frac{\rho v_i vj + p_{ij}}{r|1-M_r|} \right] dS \\
& + \frac{\partial}{\partial t} \int_S \left[\frac{\rho \mathbf{v}.\mathbf{n}}{r|1-M_r|} \right] dS
\end{aligned}
\tag{2.1}
$$

with

V = Control volume around moving body
S = Surface of body, assumed impermeable
c = speed of sound in medium
ρ = Ambient density of fluid
ρ' = Density perturbation
$\mathbf{x}$ = Observer position
t = Time
$T_{ij} = \rho v_i v_j + p_{ij} - c^2 \rho' \delta_{ij}$
= Lighthill stress tensor
$p_{ij} = p' \delta_{ij}$ + small terms dependent on viscosity
$\mathbf{y}$ = Source position
$r = |\mathbf{x} - \mathbf{y}|$
M_r = Source-observer relative velocity

$$n_i = \text{Components of surface normal on } S$$

and $[\cdot]$ implies evaluation of the quantity inside the brackets at retarded time $\tau = t - r/c$. The two surface integrals are generally considered the more important and are referred to as the *loading* and *thickness* noise contributions respectively.

2.1 Farassat's formulation

A number of more convenient formulations of equation 2.1 have been derived for the case when the solid body is a propeller. Implemented in SCRUMPI is an equivalent of Farassat's formulation 1A for subsonic propellers, [1] with the modified to include an arbitrary uniform inflow. Farassat's formulation is

$$4\pi p'_T(\mathbf{x},t) = \int_S \left[\frac{1}{1-M_r}\frac{\partial}{\partial\tau}\frac{\rho_0 v_n}{r(1-M_r)}\right] dS \tag{2.2}$$

$$4\pi p'_L(\mathbf{x},t) = \frac{1}{c}\int_S \left[\frac{1}{1-M_r}\frac{\partial}{\partial\tau}\frac{l_r}{r(1-M_r)}\right] dS + \int_S \left[\frac{l_r}{r^2(1-M_r)}\right] dS \tag{2.3}$$

For a moving source, with an inflow given by M_∞ the pressure at a stationary observer is given by (Appendix A for the derivation)

$$G(\mathbf{x},t;\mathbf{y},\tau;\mathbf{M}_\infty) = \gamma\frac{\delta(g)}{4\pi R'|1-M'_r+\gamma^2\mathbf{M}_\infty.\mathbf{M}_s|} \tag{2.4}$$

$$M'_r = -\frac{\partial R'}{\partial\tau}/c$$

(Equivalent of source relative Mach number)

$$\mathbf{M}_s = \frac{\partial\mathbf{y}}{\partial\tau}$$

(Source Mach number relative to fixed reference frame)

and the derivative conversion factor is

$$D_{t\tau} = \frac{1}{1+(\partial R'/\partial\tau)/c+\gamma^2(\mathbf{M}_\infty.\mathbf{M}_s)^2} \tag{2.5}$$

2.1.1 Use of generalised functions to represent discontinuities

In the following development, generalised functions are used to handle the source terms in the wave equation, a technique which has been used by Farassat in a number of papers, [1, 5]. The theory of generalised functions is explained by him in more detail in a NASA report, [6]. Much of the theory underlying the material in this section can be found in that report.

Generalised functions can be used in work of this kind to impose a boundary condition in a natural and convenient way. For example, a surface can be represented by a function $f(\mathbf{y}, t)$,

$$f(\mathbf{y},t)\begin{cases} < 0, & \text{inside the surface} \\ = 0, & \text{on the surface} \\ > 0, & \text{outside the surface} \end{cases}$$

and to restrict a quantity to the surface, we just multiply it by $\delta(f)$. For example,

$$\int_V Q(\mathbf{y})\delta(f)\,d\mathbf{y}$$

is equal to the integral of the function Q over the surface represented by f. In the sections below, we write the source terms for the wave equation in the form $Q\delta(f)$ where Q will be either the loading or thickness noise term. Then the source term is integrated over all time and all space to yield the noise at a time t generated by the surface source. The great advantage of this technique is that it is possible, using the rules of generalised differentiation, to develop formulations such as Farassat's formulation 3, [7], which do not have a Doppler factor $1/|1-M_r|$ in the integrand. This means that there are none of the difficulties caused by singularities which arise when the source moves transonically or supersonically.

The arrangement is shown in figure 2.1. The surface is given by $f(\mathbf{x}, t) = 0$. We also use the surface normal $\mathbf{n}$ although this could be written in terms of the gradient of the function f.

Here, we develop formulae which do have a Doppler factor. This is because formulae of the type mentioned above usually require more intense computation than the conventional form and so are best used only when required. Also formulae of the usual type are more easily interpreted and are more helpful in examining the physics of the system.

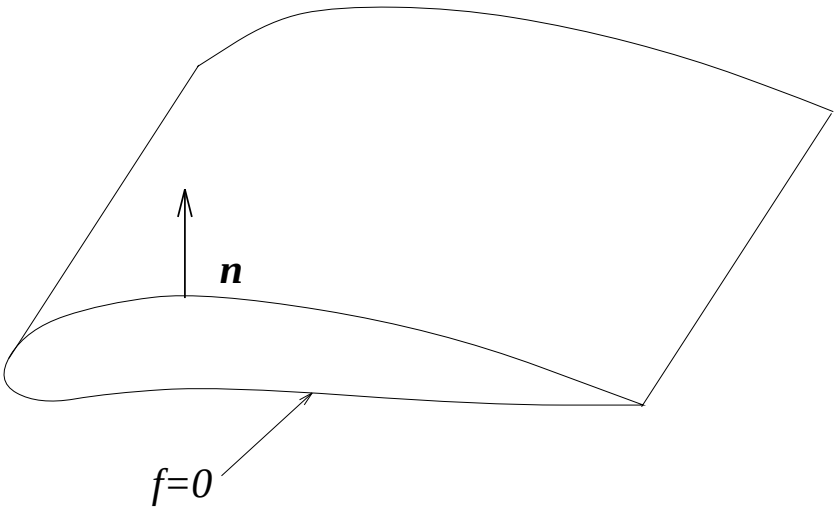

Figure 2.1: The general arrangement for solid body noise

2.2 Noise generation in a moving medium

The wave equation for a solid body moving in a uniform steady flow is now derived. The geometry for the system is shown in figure 2.1, using the surface definition $f(\mathbf{x}, t) = 0$ discussed above. A slightly more complex derivation for the case of flow along one coordinate axis can be found in Wells and Han [8]. The approach here is more general, considering the case of an inflow of arbitrary direction. The starting point is the equations of momentum and continuity (see e.g. Panton [9]) modified by multiplication with a Heaviside function of $f(\mathbf{x}, t)$ which defines the surface of the body of interest. Multiplying by $H(f)$ is simply a means of stating that the equations are valid only in the space outside the surface f. It is further required that $|\nabla f| = 1$, in accordance with the convention of Farassat [6]. The equations are, replacing $\partial/\partial t$ with $D/Dt = \partial/\partial t + u_i \partial/\partial x_i$,

$$H(f)\left(\frac{D\rho'}{Dt} + \frac{\partial}{\partial x_i}(\rho v_i)\right) = 0, \tag{2.6}$$

$$H(f)\left(\frac{D}{Dt}(\rho v_i) + \frac{\partial}{\partial x_j}(p_{ij} + \rho v_i v_j)\right) = 0. \tag{2.7}$$

These can be rewritten, since $H(f)dg/dx = \bar{d}/dx(H(f)g(x)) - g(x)\delta(f)$, where the bar on a differentiation operator signifies generalised differentia-

tion,

$$\frac{\bar{D}}{Dt}(H(f)\rho') + \frac{\bar{\partial}}{\partial x_i}(H(f)\rho v_i) = \delta(f)\frac{\partial f}{\partial x_i}[\rho_0(w_i - u_i) + \rho(v_i + u_i - w_i)], \tag{2.8}$$

$$\frac{\bar{D}}{Dt}(H(f)\rho v_i) + \frac{\bar{\partial}}{\partial x_j}(H(f)(p_{ij} + \rho v_i v_j)) = \delta(f)\frac{\partial f}{\partial x_j}(\rho v_i(u_j - w_j + v_j) + p_{ij}), \tag{2.9}$$

with w_i the velocity of the body surface. It can be seen that the delta function restricting the source quantities to the surface f as discussed in the previous section arises naturally from the use of the Heaviside function in equations 2.6 and 2.7. Then, differentiating equation 2.8 with respect to time and subtracting from it the divergence of 2.9,

$$\begin{aligned}\left(\frac{D^2}{Dt^2} - c^2\frac{\partial^2}{\partial x_i^2}\right)\rho' = &\frac{\bar{D}}{Dt}\left([\rho_0(w_i - u_i) + \rho(v_i + u_i - w_i)]\delta(f)\frac{\partial f}{\partial x_i}\right) \\ &- \frac{\bar{\partial}}{\partial x_i}\left([\rho v_i(u_j - w_j + v_j) + p_{ij}]\delta(f)\frac{\partial f}{\partial x_j}\right) \\ &+ \frac{\partial^2}{\partial x_i \partial x_j}(p_{ij} + \rho v_i v_j) - c^2\frac{\partial^2 \rho}{\partial x_i^2},\end{aligned} \tag{2.10}$$

with quantities now restricted to the support of $H(f)$ (i.e. only to be considered outside the surface f). Then, since for an impermeable surface $w_i \partial f/\partial x_i = (u_i + v_i)\partial f/\partial x_i$, equation 2.10 can be rewritten,

$$\begin{aligned}\left(\frac{D^2}{Dt^2} - c^2\frac{\partial^2}{\partial x_i^2}\right)\rho' = &\frac{\bar{D}}{Dt}[\rho_0 v_n \delta(f)] && \text{(Thickness)} \\ &- \frac{\bar{\partial}}{\partial x_i}[l_i \delta(f)] && \text{(Loading)} \\ &+ \frac{\partial^2 T_{ij}}{\partial x_i \partial x_j}, && \text{(Quadrupole)}\end{aligned} \tag{2.11}$$

where v_n is the surface normal velocity $v_i \partial f/\partial x_i$ and l_i is the force exerted by the surface on the fluid.

2.2.1 Thickness noise

The solution of the convected wave equation for thickness noise is found by convolving the source terms in equation 2.11 with the Green's function

introduced in equation 2.4,

$$4\pi p'_T = \gamma \left(\frac{\bar{\partial}}{\partial t} + u_i \frac{\bar{\partial}}{\partial x_i} \right) \int_\tau \int_{\mathbf{y}} \frac{\rho_0 v_n \delta(f)}{R'} \delta(g)\, d\mathbf{y}\, d\tau. \tag{2.12}$$

The first integration, over τ, is quite simple and is carried out in the same way as in the point-source problem in the previous chapter,

$$4\pi p'_T = \gamma \left(\frac{\bar{\partial}}{\partial t} + u_i \frac{\bar{\partial}}{\partial x_i} \right) \int_{\mathbf{y}} \frac{\rho_0 v_n \delta(f)}{R' dg/d\tau}\, d\mathbf{y}. \tag{2.13}$$

The integration over the source region, $\mathbf{y}$, is performed using the identity (Farassat [6], equation 3.80)

$$\int_{\mathbf{x}} \phi(\mathbf{x}) \delta(f)\, d\mathbf{x} = \int_{f=0} \phi(\mathbf{x})\, dS \tag{2.14}$$

with S the surface defined by $f = 0$. This yields

$$4\pi p'_T(\mathbf{x}, t) = \gamma \left(\frac{\partial}{\partial t} + u_i \frac{\partial}{\partial x_i} \right) \int_S \left[\frac{\rho_0 v_n}{R'(1 - \mathbf{M}_s.(\gamma \mathbf{R} - \gamma^2 \mathbf{M}_\infty))} \right] dS \tag{2.15}$$

where $[\cdot]$ indicates evaluation of the function inside the brackets at retarded time τ, S is the blade surface and $\mathbf{M}_s$ is the source velocity in the fixed reference frame, $\mathbf{R} = \partial R'/\partial x_i$,

$$\begin{aligned} \mathbf{R} &= (\mathbf{r} + \gamma^2 (\mathbf{M}_\infty.\mathbf{r}) \mathbf{M}_\infty)/R', \\ \mathbf{r} &= \mathbf{x} - \mathbf{y}. \end{aligned} \tag{2.16}$$

The spatial derivative can be expanded to a time derivative and the time derivatives brought under the integral sign to give a formula for thickness noise

$$\begin{aligned} 4\pi p'_T = \gamma \int_S \left[\frac{1}{1 - \mathbf{M}_s.\mathbf{D}} \frac{d}{d\tau} \left(\frac{\rho_0 v_n (1 - \mathbf{M}_\infty.\mathbf{D})}{R'(1 - \mathbf{M}_s.\mathbf{D})} \right) \right] dS \\ -\gamma c \int_S \left[\frac{\rho_0 v_n \mathbf{M}_\infty}{R'(1 - \mathbf{M}_s.\mathbf{D})} \cdot \left(\frac{\mathbf{R}}{R'} - \frac{\gamma}{c} \frac{\dot{\mathbf{R}}}{1 - \mathbf{M}_s.\mathbf{D}} \right) \right] dS, \end{aligned} \tag{2.17}$$

where the 'radiation vector', $\mathbf{D}$, is equal to $\gamma \mathbf{R} - \gamma^2 \mathbf{M}_\infty$.

2.2.2 Loading noise

The solution of the loading noise equation is

$$4\pi p'_L = -\gamma \frac{\bar{\partial}}{\partial x_i} \int_\tau \int_{\mathbf{y}} \left[\frac{\delta(g)}{R'} l_i \delta(f) \right] d\mathbf{y}\, d\tau. \tag{2.18}$$

where l_i is the ith component of the force per unit area on the fluid. This can be manipulated in the same way as the thickness noise equation above to give

$$4\pi p'_L = -\gamma \frac{\partial}{\partial x_i} \int_S \left[\frac{l_i}{R'(1 - \mathbf{M}_s.(\gamma \mathbf{R} - \gamma^2 \mathbf{M}_\infty))} \right] dS. \tag{2.19}$$

Then, expanding the spatial derivative, this becomes

$$\begin{aligned} 4\pi p'_L = {} & \frac{\gamma}{c} \int_S \left[\frac{1}{1 - \mathbf{M}_s.\mathbf{D}} \frac{d}{d\tau} \left(\frac{\mathbf{l}.\mathbf{D}}{R'(1 - \mathbf{M}_s.\mathbf{D})} \right) \right] dS \\ & + \gamma \int_S \left[\frac{\mathbf{l}}{R'(1 - \mathbf{M}_s.\mathbf{D})} . \left(\frac{\mathbf{R}}{R'} - \frac{\gamma}{c} \frac{\dot{\mathbf{R}}}{1 - \mathbf{M}_s.\mathbf{D}} \right) \right] dS. \end{aligned} \tag{2.20}$$

2.2.3 Field components

Equations 2.17 and 2.20 can be expanded into the near- and far-field terms. The thickness noise equation becomes

$$\begin{aligned} p'_T = {} & \int_S G \frac{1 - \mathbf{M}_\infty.\mathbf{D}}{1 - \mathbf{M}_s.\mathbf{D}} \left(\rho \dot{v_n} + \gamma \rho v_n \frac{(\dot{\mathbf{M}}_s.\mathbf{R} + \mathbf{M}_s.\dot{\mathbf{R}})}{1 - \mathbf{M}_s.\mathbf{D}} \right) dS, \quad \text{(Far field)}, \\ & - c \int_S G \frac{\rho v_n}{R'} (\mathbf{M}_\infty.\mathbf{R} - \mathbf{M}_s.\mathbf{R}(1 - \mathbf{M}_\infty.\mathbf{D}))\, dS, \qquad \text{(Near field)}, \end{aligned} \tag{2.21}$$

and the loading noise terms expand to

$$\begin{aligned} p'_L = {} & \frac{1}{c} \int_S \frac{G}{1 - \mathbf{M}_s.\mathbf{D}} \left(\gamma \mathbf{l}.\mathbf{D} \frac{\dot{\mathbf{M}}_s.\mathbf{R} + \mathbf{M}_s.\dot{\mathbf{R}}}{1 - \mathbf{M}_s.\mathbf{D}} + \dot{\mathbf{l}}.\mathbf{D} \right) dS, \quad \text{(Far field)}, \\ & + \int_S \frac{G}{R'} (\mathbf{l}.\mathbf{R} + \mathbf{l}.\mathbf{D}\mathbf{M}_s.\mathbf{R})\, dS, \qquad \text{(Near field)} \end{aligned} \tag{2.22}$$

where G contains the Green's function terms,

$$G = \frac{\gamma}{4\pi R'(1 - \mathbf{M}_s.\mathbf{D})}.$$

Chapter 3

Implementation

General issues of implementation and operation are considered in this chapter. In general, the numerical routines used are based on the wise words of Press et. al. in Numerical Recipes, [10], although I haven't used their routines so that the code can be freely redistributed.

3.1 Geometry

The geometric definitions are as follows. The propeller geometry is defined in a righthanded frame with the three basis vectors $\hat{\mathbf{t}}$, $\hat{\mathbf{n}}$ and $\hat{\omega}$. $\hat{\omega}$ is aligned with the axis of rotation of the propeller. $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ are set so that when $\hat{\omega}$ is aligned with the z-axis of the global coordinate system, they lie along the x and y axes respectively. $\hat{\omega}$ is specified in the global coordinate system, $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ rotate with the blade.

The observer position is specified in the global coordinate system as is the propeller hub position. The hub position is specified as the $t = 0$ initial position and a translational velocity.

3.2 General issues

In developing the code, the main aim has been to allow flexibility in geometry (observer and propeller position, for example) and operating conditions (flow direction say). This has meant trading off some efficiency.

To allow for this flexibility, Cartesian coordinates are used throughout with vector notation and operations performed in these coordinates, rather than in a cylindrical coordinate space as is more common. The main data type used is called `vector` which is just a `struct` containg three floating

point numbers, `x1`, `x2` and `x3`. Two operations are defined to calculate dot and cross (vector) products of these types.

3.3 Program flow

The general flow of the program is as follows, see the source code for more details of implementation.

1. Set default values for various geometric and dynamic variables and the flow conditions (c, p_0 and ρ among others).

2. Read observer position from the command line.

3. Read the operating data

   (a) Read the data header, which contains the data specified in section B.

   (b) Start reading the blade geometry data.

   (c) Allocate storage for blade geometry and aerodynamic data.

   (d) Read the blade nodal position data followed by the pressure and velocity data.

4. Set up the geometric data.

   (a) Set up the list of nodes for each element.

   (b) Calculate the surface normal for each node.

5. Calculate the noise for each time step.

   (a) Step over the specified time range.

   (b) At each time point step through the list of nodes and calculate the loading and thickness noise functions for each.

   (c) Step through the elements and integrate over each one to find the total loading and thickness noise.

6. Shift and sum the time record to find the net time record for the whole propeller.

3.3.1 Noise calculation

Calculation of the noise from a node proceeds as follows.

1. Calculate the basic variables for the node at time t. These are G, τ, $\partial G/\partial\tau$, $\mathbf{x}_{\text{observer}} - \mathbf{x}_{\text{source}}$, R', $dR'/d\tau$, $\mathbf{M}_s$ and $\dot{\mathbf{M}}_s$.

2. Calculate the blade surface properties at the node at time τ, these are p, $dp/d\tau$, v_n and $dv_n/d\tau$.

3. Calculate the loading and thickness noise contributions for the node.

   (a) The loading noise is calculated by finding the current value of the surface normal, and then calculating the contribution from the node using the formulation detailed in chapter 2.

   (b) The thickness noise requires no further calculation and the formulation of chapter 2 can be applied immediately.

The integration to find the noise from an element is performed analytically using the formula given by Gere and Timoshenko for integration over a triangular element. It is assumed that the loading and thickness noise contribution varies linearly over a triangular patch. I may or may not decide to do something more sophisticated in a future version.

3.3.2 Element list generation

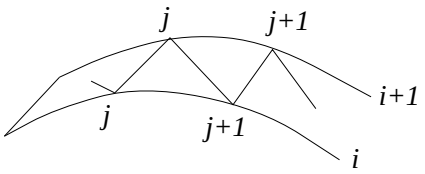

Figure 3.1: Element setup conventions

Figure 3.1 shows the assignment of nodes to the element shown. For the area bounded by the ith and $i+1$th blade sections, and the jth and $j+1$th points of each of those sections, the two traiangular elements formed are $< (i,j)\,(i+1,j+1)\,(i+1,j) >$ and $< (i,j)\,(i,j+1)\,(i+1,j+1)$ as shown in figure 3.2. Note that the ordering of nodes in an element is anticlockwise for both elements.

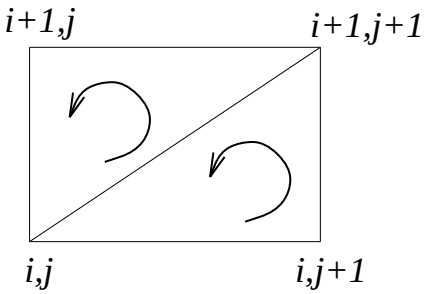

Figure 3.2: Nodal ordering convention

3.3.3 Nodal normal generation

Normals are calculated for each node by taking the cross product of the vectors from a node to its neighbours. (This is not a very clean way of doing things and should really be improved, but it'll have to wait until I get around to it.) Figure 3.3 shows the setup for finding a normal for node (i, j). $\mathbf{x}$

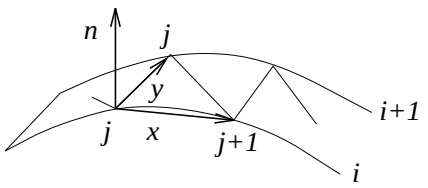

Figure 3.3: Nodes for calculation of nodal normal

is the vector from node (i, j) to node $(i, j + 1)$ and $\mathbf{y}$ the vector from node (i, j) to node $(i + 1, j)$. Then the unit normal $\hat{\mathbf{n}}_s$ is

$$\hat{\mathbf{n}} = \frac{\mathbf{x} \times \mathbf{y}}{|\mathbf{x} \times \mathbf{y}|}$$

For the nodes at the blade tip the vectors are calculated moving inboard—i.e. the $(i - 1, j)$ and $(i, j + 1)$ nodes are used instead.

3.4 Output data

3.4.1 Acoustic data

The acoustic data output are

- loading noise for one blade.

- loading noise for B blades.
- thickness noise for one blade.
- thickness noise for B blades.

3.4.2 Retarded shape

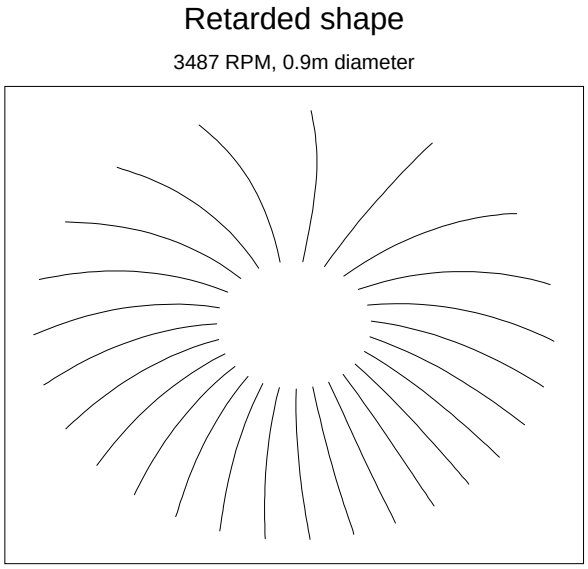

Figure 3.4: Retarded shape of leading edge for subsonic prop

The user can select an option to write a "retarded shape" to file as the calculation proceeds. This is a record of the position of a set of points at the retarded time for selected time steps. It's like looking at the blade with light that travels at the speed of sound. In SCRUMPI the shape of the blade and some other parameters can be written to a file (or to standard output) at selected time steps. The retarded shape can give a useful insight into the physics, especially for high speed rotors, by relating the retarded time back to the geometry of the system in the time domain. Using the `-r` option the user specifies how often the retarded shape should be written to `stdout`. If 0 is specified the retarded shape is not written. The option `-R`*filename* allows the user to specify a filename for the data. There is a postprocessor (see chapter 5) which will process the retarded data files and generate a `geomview` file for visualisation purposes.

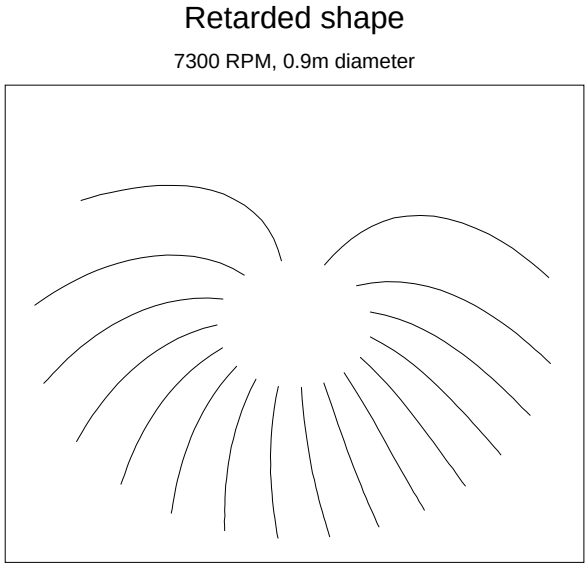

Figure 3.5: Retarded shape of leading edge for transonic prop

Figures 3.4 and 3.5 show retarded shapes for the leading edge of a blade operating subsonically and transonically for an observer in the near field. The blade positions in figure 3.4 are more or less evenly spaced as you might expect for a subsonic propeller but in figure 3.5 there is a point, about ninety degrees before the observer position, where the blade positions "separate". This corresponds to the point of maximum blade velocity towards the observer.

Chapter 4

Hacking SCRUMPI

Bizarre as it may seem, there are some people who might want to predict the noise from something which does not go round and round. In the interests of such poor lost sheep, this good shepherd provides some hints on how to go about such a task with the minimum of extra programming.

4.1 How to use SCRUMPI for unnatural purposes

The steps are fairly simple.

1. You will need to alter the functions in the file `kinematics.c` to return the position and velocity of a mesh point. The operations are all in Cartesian coordinates so it shouldn't be too difficult. Some day I may even get around to writing all this as a library to be linked to user-supplied functions but don't expect anything too soon.

2. Recompile SCRUMPI.

3. Do the aerodynamic calculation (you'll have to use your own facilities to do this).

4. Set up the data file as shown in appendix B and feed it to SCRUMPI.

5. Try to work out what the answer means.

Remember that SCRUMPI doesn't, as yet, handle supersonic sources so don't try to work out the noise from a speeding bullet or an engineer on his way to the bar. The author would be interested in hearing from anyone who uses SCRUMPI like this and actually gets some useful results out.

4.1.1 Setting the source position

In the function `SourcePosition`, the position of the current node (specified by the global variables `section`, `node`) is found for time `time` (an input parameter). You can set this any way you like (it doesn't even have to change with time). In SCRUMPI this function sets some other variables which are used by the velocity calculation function but there is no reason why you would have to use them and there is no reason why you can't do some other manipulations as well.

4.1.2 Setting the source velocity

There is also a function called `SourceVelocity` where the velocity and acceleration of the current node are set. This is a fairly simple task in SCRUMPI as most of the hard work has been done in the source position calculation. This may vary for you.

4.1.3 Setting the source axes

There is also a function in `kinematics.c` called `SetAxes`. This is called at the start of the program to set up the source-centred coordinate frame. In SCRUMPI there are three variables called `OmHat0`, `THat0` and `NHat0` ($\hat{\omega}_0$, $\hat{\mathbf{t}}_0$ and $\hat{\mathbf{n}}_0$). These are rotated by `SourcePosition` to generate the unit vectors $\hat{\omega}$, $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ which are used to convert blade-fixed coordinates to global coordinates. If your source doesn't rotate then you may well want to set $\hat{\omega}$, $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ at the start of the program and forget about them after that.

4.2 Train I ride: An "example"

As an example of how you might go about this kind of thing, consider the case of a train going round a bend[1]. A blues player in one of the wagons, wheezing through his old harmonicker is, somewhat implausibly, audible to the depot agent standing next to the line. How would you predict the sound he hears?

4.2.1 Source position

This is a doddle. You know the geometry of the track so you know the path followed by the instrument. If you wanted you could include a calculation for

[1] I haven't actually done this, but this is how I would go about the problem

the reeds of the harmonica, but their displacement would be so small that it is irrelevant. For the distance calculation, all you need to worry about is the distance to the instrument itself.

4.2.2 Source velocity

This is where the reed vibration comes in. The velocity is another easy calculation. Start with the reed velocity which is a simple periodic function. It's just a sum of a few sinusoids at a number of natural frequencies. You could assume that all displacement is vertical (few musicians lie on their side to play) and then there is only a z velocity to worry about. Then just add on the train velocity. Everything is in Cartesian coordinates so it shouldn't be too hard.

The only tricky part is including out the reed surface pressure. This will vary periodically and you have to make sure that the coefficients of the Fourier series for the pressure at each node have the right phase so that the pressure and reed velocity correspond.

Then just run the program. Easy, isn't it?

Chapter 5

Postprocessing with scrapple

There is a simple postprocessing program included with SCRUMPI, currently called SCRAPPLE which will take a retarded-time output file and generate an OOGL file (OOGL is the language used by `geomview`) for visualisation. SCRAPPLE takes the retarded time data and generates a list of meshes, one for each time step, colouring the nodes according to the value of some variable at that node (this would be one of the variables which is stored by SCRUMPI, Green's function or Mach number, for example).

To use SCRAPPLE, you will also need `geomview`, a package for displaying three dimensional geometric data. It is available from the Geometry Centre and can be obtained at the ftp address `ftp.geom.umn.edu` or via the world wide web, from `http://www.geom.umn.edu/`. It's a dead good piece of software and well worth having for other purposes (waking you up with a cup of tea and a kiss in the morning, that kind of thing).

5.1 Usage

At the moment SCRAPPLE is quite a simple program. It is invoked by typing

`scrapple` *(options) (input filename)*

The available options are shown in table 5.1. The input file is one generated using the retarded data options of SCRUMPI (§3.4.2).

If no input filename is specified, SCRAPPLE reads from standard input, and if no output filename is passed using the `-o` option, then output goes to standard output.

The `-f` option is used to specify the index of the variable which will be used to colour the mesh nodes. The default value is 1; the first variable will be used.

-a	write ASCII data ASCII	-b	write binary data (**default**)
-f#	field to process (**1**)	-h	print help message
-o*	output filename (**standard output**)	-q	run quietly (**print messages**)
-W	print (no) warranty message		

Table 5.1: SCRAPPLE options: defaults are shown **bold**.

When you have the OOGL file generated by SCRAPPLE (`pressure.out`, say), just run `geomview` over it,

```
geomview pressure.out
```

or run `geomview` and select the output file on the menu.

5.2 Operation

SCRAPPLE works quite simply. It reads the retarded time position and property data from the file written by SCRUMPI and stores them in memory. It then scales the required data to lie between 0 and 1. This scaled value is then used as the 'hue' part of a HSV colour map. The colour values are converted to RGB format for the benefit of `geomview` and written, with the mesh point positions to the output OOGL file.

At the moment only the OOGL (`geomview`) format is handled and since `geomview` is free software, it is unlikely that any other system will be catered for in the near future. Unless someone has some very convincing reason for wanting it done.

Appendix A

Derivation of sound from a point source moving in a flow

The Green's function required for moving source acoustics with an arbitrary uniform inflow is readily derived by adapting a well-known Green's function for the sound radiated from a source in a uniform flow to the problem of sound radiated from a source moving in a flow. This proceeds in two stages,

1. the adaptation of the Green's function to a set of rotated axes, section A.1.

2. the development of this new Green's function to include source motion effects, section A.1.1.

A uniform, steady flow is the simplest case of fluid motion imaginable. It is justified on the grounds that

- without information about the large scale flow field around a body, any more detailed theory is useless anyway.

- outside a boundary layer, it seems to be an acceptable approximation to the flow around a propeller. In particular, it seems a reasonable assumption for modelling convective effects.

A.1 The sound from a source moving in a uniform flow

The starting point for this derivation is the Green's function given by Lakhtakia et. al., [11] for sound radiated in a uniform flow, with a source at $\mathbf{y}$, an ob-

server at $\mathbf{x}$ and a flow of Mach number M_∞ in the x_3 direction,

$$G(\mathbf{x},t;\mathbf{y},\tau) = \gamma\frac{\delta(\tau - t + R'/c - \gamma^2 M_\infty(x_3 - y_3))}{4\pi R'} \tag{A.1}$$
$$R' = [(x_1-y_1)^2 + (x_2-y_2)^2 + \gamma^2(x_3-y_3)]^{1/2}$$
$$\gamma^2 = \frac{1}{1-M_\infty^2}$$

which has the same form as the usual $1/R$ Green's function except that the "effective distance" between source and observer has changed and there is an additional retarded time term.

Equation A.1 can be readily adapted to the general case of a flow with Mach number vector $\mathbf{M}_\infty$,

$$G(\mathbf{x},t;\mathbf{y},\tau) = \gamma\frac{\delta(\tau - t + R'/c - \gamma^2\mathbf{M}_\infty.(\mathbf{x}-\mathbf{y}))}{4\pi R'} \tag{A.2}$$
$$R' = (|(\mathbf{x}-\mathbf{y})|^2 + (\gamma\mathbf{M}_\infty.(\mathbf{x}-\mathbf{y}))^2)^{1/2}$$
$$\gamma^2 = \frac{1}{1-|\mathbf{M}_\infty|^2}$$

A.1.1 Source motion

To include source motion effects, we can proceed in the usual fashion for this problem, see e.g. Dowling and Ffowcs Williams, [12]. The sound from a source distribution over the volume V, is given by

$$p'(\mathbf{x},t) = \int_{-\infty}^{\infty}\int_V G(\mathbf{x},t;\mathbf{y},\tau)s(\mathbf{y},\tau)\,dV\,d\tau \tag{A.3}$$

(A.4)

In this case,

$$p'(\mathbf{x},t) = \int_{-\infty}^{\infty}\int_V \gamma\frac{\delta(g)}{4\pi R'}\,dV\,d\tau \tag{A.5}$$
$$g = \tau - t + R'/c - \gamma^2(\mathbf{M}_\infty.(\mathbf{x}-\mathbf{y}))/c$$

The delta function allows us to perform the integration over V immediately and we have, from generalised function theory, the result

$$\int_{-\infty}^{\infty} f(x)\delta(g(x))\,dx = \left[\frac{f(x)}{|dg/dx|}\right]_{x=x^*}$$

where the notation $[\cdot]_{x=x^*}$ denotes evaluation of the term in brackets at the roots of $g(x)$, $x = x^*$. If $g(x)$ has more than one root then the term is evaluated at each of these roots and summed.

This gives us for the sound from a moving point source in a flow $\mathbf{M}_\infty$

$$G(\mathbf{x}, t; \mathbf{y}, \tau; \mathbf{M}_\infty) = \gamma \frac{\delta(g)}{4\pi R' |1 + \partial R'/\partial\tau/c + \gamma^2 \mathbf{M}_\infty . \dot{\mathbf{y}}/c|} \tag{A.6}$$

with the assumption that the observer does not move.

This equation can then be modified to bring it into line with the more usual form for a moving source,

$$G(\mathbf{x}, t; \mathbf{y}, \tau; \mathbf{M}_\infty) = \gamma \frac{\delta(g)}{4\pi R' |1 - M'_r + \gamma^2 \mathbf{M}_\infty . \mathbf{M}_s|} \tag{A.7}$$

$$M'_r = -\frac{\partial R'}{\partial \tau}/c$$

(Equivalent of source relative Mach number)

$$\mathbf{M}_s = \frac{\partial \mathbf{y}}{\partial \tau}$$

(Source Mach number relative to fixed reference frame)

Appendix B

Input data

The input data to SCRUMPI are as follows. All data can be in one file or in separate files. The ordering is important though. All input files are ASCII. A binary file format is planned but there are more important things to worry about for the minute. In any of the input files, except the header, lines beginning with the character # are ignored and can be used as comments.

B.1 Header data

The header contains the main operating parameters in this order

- A text string.
- The inflow velocity vector in metres per second.
- The speed of sound in metres per second.
- The ambient pressure in Pascals.
- The ambient temperature in Kelvin.
- The propeller rotational velocity, RPM.
- The direction of rotation, ± 1, positive for anti-clockwise rotation.
- The number of blades.
- The blade pitch in degrees.

For example,

```
A_COMMENT
0.0 0.0 229.211
327.4444
100000.0
293.0
3487.0
-1
6
61.79999
```

specifies a 6-bladed propeller operating at 3487 rpm, in air at 100000 Pa and 293 K. Speed of sound is 327 m/s and the inflow is axisymmetric at 229 m/s.

B.2 Geometric data

The geometry of the blade is specified in Cartesian coordinates in a coordinate system whose z-axis lies along the axis of rotation of the propeller. The format is as follows.

- The number of blade sections specified.
- The number of nodes per section.
- The nodal positions, a section at a time with the last point of each section identical to the first, to guarantee closure. All dimensions are in metres.

For example

```
# These are the geometry data
 21
 73
 5.16021E-03 .1337101 4.77570E-02
 4.78180E-03 .1336599 4.72610E-02
```

is part of a file specifying a blade with 21 sections and 73 points per section.

B.2.1 Point-force geometry

When SCRUMPI is being run in point-force mode the geometry specification has to change slightly. You need to specify a *normal* (i.e. the direction of the force) as well as its magnitude. For example—

```
# point force geometry definition
1
1
0.3 0.0 0.0

# the force direction
0.707 0 -0.707
```

Here the point force is concentrated at $(0.3, 0, 0)$ (all in meters) and the direction of the force is $(\sqrt{2}, 0, -\sqrt{2})$ (negative because the thrust is upstream).

B.3 Pressure data

The blade surface pressure at each mesh node can be specified in either the time or frequency domain. A frequency domain specification is preferred and is the default option but the time domain form can be useful on occasion

B.3.1 Frequency domain loading

The pressure data are specified in the form of spectra with the frequency, magnitude and phase of each component specified. Each node's spectrum gets one line in the input file with nodes in the same order as in the geometry file. Each line has the format,

(frequency, magnitude, phase) ... (frequency, magnitude, phase)

For example,

```
# This is the frequency domain pressure data
 0.0 7.2892920e+04 0.0 5.8116667e+01 3.6446460e+03 213
```

is a line specifying that the node has a pressure given by

$$p_{\text{node}}(t) = 7.289 \times 10^4 + 3.644 \times 10^3 \cos(2\pi 58.1 t + 213^\circ) \tag{B.1}$$

All frequencies are specified in Hz with phases given in degrees. Magnitudes are given in Pa.

B.3.2 Time domain loading

When the loading data are specified in the time domain, the data on each line are interpreted as a list of pressures sampled at evenly spaced points in time, starting at time 0s. The command line switch `-t` is used to specify the sampling rate for these points and the switch `-L` specifies the length of the record. Note that for retarded times outside the range of the supplied time record a pressure of zero is used—there is no 'wrap-around'. Calculation of the pressure and its derivative are performed using linear interpolation.

B.4 Velocity data

Optionally, a surface normal velocity distribution can be specified (use the `-V` command line option). Then the velocities can be specified in the same way as the pressures in §B.3 in either the time or frequency domain. If the loading is specified in the time domain the velocities should be also and similarly for the frequency domain. Note that the velocities will be ignored if the `-V` command line option is not used.

Appendix C

Installation on various platforms

C.1 General notes

Installing SCRUMPI should be a doddle but probalbly won't be. It is designed to be compiled in the following fashion—

1. Edit the Makefile. You will need to set the name of the compiler you're using (`cc` or `gcc` are the usual ones) and also the type of system. Just uncomment the line with your system's name on it and comment out every other machine. If your machine isn't listed in the Makefile, then try `GENERIC`. If that doesn't work, you will probably have to hack `config.h`.

   You can also decide to use single or double precision floating point operations by uncommenting the line which defines `DOUBLE`.

2. You may want to hack `config.h`. This contains settings for the various defaults (observer position, for example) as well as the `#include`s which vary from machine to machine.

3. Type `make` and sit back.

C.2 Different platforms

SCRUMPI should install easily enough on a standard Un*x system. Some machines require that the code be tweaked slightly, however.

C.2.1 MIPS

MIPS `cc` does not support the '%g' format specifier in `scanf`. This allows `scanf` to read in either floating point or exponential format. If you are compiling under MIPS you need to do a search and replace and change each occurence of '%g' to read '%e' and then make sure that all your floating point input data are in exponential format (this is a good idea anyway). You could change to '%f' and read your data in floating point but exponential is safer. The affected files are `files.c` and `main.c`.

C.2.2 Windows 95/NT

SCRUMPI can be compiled and run under Windows 95/NT if you install the Cygwin 32 development environment. This gives you a Unix-like shell which will allow you to compile SCRUMPI using `gcc` and then run it. You can get Cygwin from `http://www.cygnus.com/`.

Bibliography

[1] F. Farassat. Linear acoustic formulas for calculation of rotating blade noise. *AIAA Journal*, 19(9):1122–1130, September 1981.

[2] F. Farassat. A new aerodynamic integral equation based on an acoustic formula in the time domain. *AIAA Journal*, 22(9):1337–1340, September 1984.

[3] L. Gutin. On the sound field of a rotating propeller. Technical Memorandum 1195, NACA, Langley Aeronautical Laboratory, Langley Field, Va. USA, 1948.

[4] J. E. Ffowcs Williams and D. L. Hawkings. Sound generation by turbulence and surfaces in arbitrary motion. *Phil. Trans. Roy. Soc. Lond. A.*, 264:321–342, 1969.

[5] F. Farassat. Theoretical analysis of linearized acoustics and aerodynamics of advanced supersonic propellers. In *Aerodynamics and acoustics of propellers*, volume 10. AGARD, October 1985.

[6] F. Farassat. Introduction to generalized functions with applications in aerodynamics and aeroacoustics. NASA Technical Paper TP 3428, NASA, National Aeronautics and Space Administration, Langley Research Center, Hampton, Virginia 23681-0001, May 1994.

[7] F. Farassat. Prediction of advanced propeller noise in the time domain. *AIAA Journal*, 24(4):578–584, 1986.

[8] V. L. Wells and A. Y. Han. Acoustics of a moving source in a moving medium with application to propeller noise. *Journal of Sound and Vibration*, 184(4):651–663, 1995.

[9] R. L. Panton. *Incompressible Flow.* John Wiley & Sons, Inc., 1984.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[11] Akhlesh Lakhtakia, Vijay K. Varadan, and Vasundra V. Varadan. Green's functions for propagation of sound in a simply moving fluid. *J. Acoust. Soc. Am.*, 85(5):1852–1856, 1989.

[12] A. P. Dowling and J. E. Ffowcs-Williams. *Sound and Sources of Sound.* Butterworth, 1983.