# SPRINT2D Software for Convection Dominated PDEs

**M.Berzins**
**S.V.Pennington**
**P.R.Pratt**
**J.M.Ware**

ABSTRACT
SPRINT2D is a set of software tools for solving time-dependent partial differential equations in two space variables. The software uses unstructured triangular meshes and adaptive error control in both space and time. This chapter describes the software and shows how the adaptive techniques may be used to increase the reliability of the solution procedure for a challenging combustion problem. The recent construction of a problem solving environment (PSE) has partially automated the use of SPRINT2D. This PSE consists of tools such as a visual domain specification tool, which helps ease the input of complex geometries, and a visual problem specification tool. After describing these components an evaluation will be made of SPRINT2D and its associated PSE.

## 1 Introduction

Two important trends in the development of numerical software for partial differential equations are to make numerical methods more reliable through the use of adaptive error control, and to make the software easier to use and its results easier to understand by embedding it in an interface layer. The combination of this layer and the core numerical software is classified as a Problem Solving Environment (PSE). The form of such a PSE is by no means clear; PSEs have been defined as capable of solving problems by communicating in the user's own terms, [6]. Stetter, [13], looks at the wide variety of scientific tools currently available and advocates the utilisation and integration of such tools to form PSEs devoid of explicit programming. A PSE can therefore be viewed as a collection of tools that provide a bridge between the problem the user wishes to solve and scientific software. The aim of a PSE is that it should enhance the solution process for example by increasing the reliability of the solution or decreasing the time spent from specification to solution. Adjerid and Flaherty et al. [1] identify eight components of a successful PSE:

- A computer algebra interface to describe the PDE and data.
- A geometric modelling system to describe the domain.
- A mesh generator to create a computational mesh.
- A numerical solution procedure to solve the PDE.
- Error estimation procedures to give accuracy measures.
- Adaptive strategies to improve solution resolution when needed.
- Parallel solution capabilities for increased performance when needed.
- Visualisation tools to analyse and interpret results.

The aim of this chapter is to address the twin themes of mesh adaptation and PSEs [10] in connection with the SPRINT2D software for convection-dominated PDEs [14]. Sections 2 and 3 will consider the SPRINT2D numerical PDE code. Section 4 looks at the software tools that are part of the PSE surrounding the numerical code. Section 5 contains two case studies, including and engineering example from combustion modelling, to show how the software functions. Finally Section 6 evaluates the success of the approach taken.

## 2 The SPRINT2D Software

The SPRINT2D software [14] solves time-dependent partial differential equations by using the method of lines to discretise in space thus reducing the PDEs to a system of ODEs (Ordinary Differential Equations) which can then be integrated using existing software packages. This separation of space and time and the use of ODE software makes it possible to combine different combinations of spatial and temporal discretisation as required. SPRINT2D uses a cell-centred finite volume method in which the PDE is integrated over an element and the divergence theorem applied to replace the area integral for the fluxes by a line integral around the edge of the element. The flux functions in the PDE are then used to calculate the numerical flux between adjoining elements. Although the finite volume method may use any form of spatial elements, the use of triangular elements allows complex domains to be modelled, and when used in conjunction with spatial adaptivity provides a powerful modelling environment, [4]. This is particularly true when temporal local error control and spatial error estimation and control are used. The PDEs solved by SPRINT2D are:

$$\beta \frac{\partial \underline{U}}{\partial t} + \frac{\partial}{\partial x}\underline{f}^x + \frac{\partial}{\partial y}\underline{f}^y = \frac{\partial}{\partial x}\underline{g}^x\left(\frac{\partial \underline{U}}{\partial x}, \frac{\partial \underline{U}}{\partial y}\right) + \frac{\partial}{\partial y}\underline{g}^y\left(\frac{\partial \underline{U}}{\partial x}, \frac{\partial \underline{U}}{\partial y}\right) + \underline{S} \quad (1.1)$$

where all the functions $\beta, \underline{f}^x, \underline{f}^y, \underline{g}^x, \underline{g}^y$ and $\underline{S}$ are allowed to depend on $\underline{U}$ , $x$ and $t$. For steady problems $\beta$ is set to zero. The convective fluxes $\underline{f}^x$ and $\underline{f}^y$ may give rise to wave-like structures in the solution $U$, and the terms $\underline{g}^x$ and $\underline{g}^y$ define the diffusive fluxes. The source term $S$ can be used to add other processes such as reaction terms including chemical kinetics. The three types of boundary conditions allowed by the package are Dirichlet, Neumann and flux conditions in which the solution, normal derivatives and fluxes are specified. Examples of the types of PDE problems that SPRINT2D has been applied to are given in [3] and include atmospheric dispersion problems, shallow water equations, combustion problems and gas jets. A taxing combustion modelling problem is considered in Section 5.

## *2.1 An Overview of SPRINT2D*

The SPRINT2D software has the layered structured shown in Figure 1. The top layer of the software consists of the PSE type interface tools- the Visual Display Tool (VDS) and the Visual Problem Specification Tool (VPS). An important part of the specification process for solution of two space dimensional PDEs is the definition of the region over which the problem is to be solved. Once this is done, either by hand or by the VDS tool, this region can then be meshed to provide a suitable triangulation of the domain for the numerical solution process. There are currently two mesh generation software packages that can be used by SPRINT2D; the KSLA mesh generator [5] and the GEOMPACK mesh generator [7]. Once the mesh has been generated the user must either use the VPS tool to create a C driver for SPRINT2D or write a driver program by hand. The SPRINT2D package is implemented on top of two existing numerical packages: SPRINT and NAESOL - all the codes being written in C. After applying spatial discretisation to time-dependent problems, the resulting system of ODEs is integrated in time by the SPRINT integrators.

Spatially discretising steady problems results in a system of non-linear equations which are solved by the non-linear solver package NAESOL, [12], but are not considered further here. The modular nature of the software allows additional solution modules to be added to the package. The main options are: dense sparse and iterative linear algebra modules; an operator splitting module; and Theta and Backward Differentiation Formula time integration methods. The TRIAD package provides the routines to perform any spatial adaptivity using $h$–refinement.

The C driving program for SPRINT2D specifies the PDEs and the solution techniques to be used. The first part of the driver program must include the relevant header files for the SPRINT2D package and modules that are to be used. The user, in the driver program, needs to specify the following information: a file containing a specification of the physical domain; relative and absolute tolerances for the adaptivity routines; a Riemann solver for

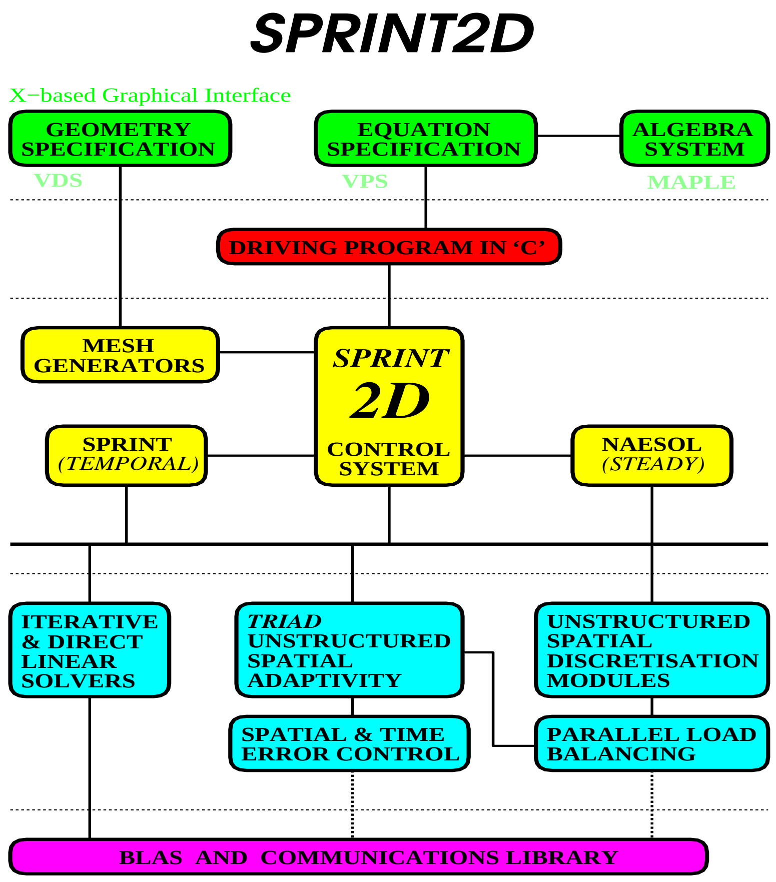

SPRINT2D
X−based Graphical Interface
GEOMETRY SPECIFICATION
VDS
EQUATION SPECIFICATION
VPS
ALGEBRA SYSTEM
MAPLE
DRIVING PROGRAM IN 'C'
MESH GENERATORS
SPRINT 2D CONTROL SYSTEM
SPRINT (TEMPORAL)
NAESOL (STEADY)
ITERATIVE & DIRECT LINEAR SOLVERS
TRIAD UNSTRUCTURED SPATIAL ADAPTIVITY
UNSTRUCTURED SPATIAL DISCRETISATION MODULES
SPATIAL & TIME ERROR CONTROL
PARALLEL LOAD BALANCING
BLAS AND COMMUNICATIONS LIBRARY

FIGURE 1. Outline of Sprint Software

the advective fluxes $\underline{f}^x$, $\underline{f}^y$ (see Section 2.2); a flux function for $\underline{g}^x$ and $\underline{g}^y$; a source term function $\underline{S}$; boundary conditions; and initial conditions. These functions have to follow a fixed interface in returning values to SPRINT2D.

## *2.2 Riemann Solvers and Boundary Conditions*

For convection-dominated PDEs, correct specification of the flux and careful space discretisation are essential to avoid unphysical oscillations in the discrete solution. A standard approach to ensure a stable solution is to place more emphasis on the information coming from the direction of the flow (the upwind values) in discretising the advective parts of the PDE. In order to maintain stable solution values upwinding is combined with a non-linear scheme that changes order by limiting the numerical flux that passes between cells, see [4, 9]. For simple problems, e.g. linear advection, the choice of upwinding direction is obvious. However, for complex systems, the direction may alternate, or a combination of both left and right values maybe needed for a system of PDEs, see [9]. In this case an approximate (sometimes exact) Riemann solver is used to calculate the advective flux in the code using a combination of knowledge about the PDE and left and right solution values, [9]. For example in the case of the knock problem defined in Section 5.2 below, the fluxes $\underline{f}^x$ and $\underline{f}^y$ must be calculated given left $\underline{U}_L = (\rho, \rho u, \rho v, E, \rho z)^T_L$ and right $\underline{U}_R = (\rho, \rho u, \rho v, E, \rho z)^T_R$ solution values at the midpoint of each edge. This calculation is a nontrivial task, see [4]. The Riemann solver is used to implement flux or derivative boundary conditions. For example, the reflective boundary conditions in Section 5.2 are imposed by setting the exterior 'normal' velocity to be the opposite sign to the normal velocity at the boundary from the interior. The values of all the other variables on the 'exterior' being the same as the interior values. All other 'outside' solution values are the same as the interior values. Although it may be possible to construct approximate Riemann solvers automatically, there are many situations in which the user must specify the Riemann solver.

## 3 Mesh Generation and Adaptivity

The main attraction of unstructured triangular meshes is that they can approximate arbitrary domains more easily than quadrilateral based meshes. The initial meshes used in SPRINT2D are created from a geometry description using the KSLA [5] or GEOMPACK [7] mesh generators. GEOMPACK constructs the mesh by decomposing the input geometry into simpler polygons and then meshing these polygons. As a semi–automatic mesh generator GEOMPACK requires additional information at the beginning of the specification file to accomplish this. This information provides the user with the ability to control various aspects of the final mesh such as desired number of triangles, mesh smoothness and the way in which the geometry is decomposed into simpler polygons. These meshes are then refined and coarsened by the TRIAD [14] adaptivity module which uses data structures to enable efficient mesh adaptation. For the $i$th PDE component on the $j$th triangle, a local error estimate $e_{i,j}(t)$ is calculated from the difference between the solution using a first order method and that using a second order method, see [4] for details. For time dependent PDEs this estimate shows how the spatial error grows locally over a time step. A refinement indicator for the $j$th triangle is defined by an average scaled error ($serr_j$) measurement over all $npde$ PDEs

$$serr_j = \sum_{i=1}^{npde} \frac{e_{i,j}(t)}{atol_i / A_j + rtol_i \times u_{i,j}}, \tag{1.2}$$

where *atol* and *rtol* are the user-supplied absolute and relative error tolerances. This formulation for the scaled error provides a flexible way to weight the refinement towards any PDE error. An integer refinement level indicator is calculated from this scaled error to give the number of times the triangle should be refined or derefined.

In the refinement case, all the neighbouring triangles which share an edge with a refined triangle are refined towards that shared edge. Similarly, all triangles with a vertex in common with the original triangle are refined towards that vertex. Finally Bank's green rule is applied to ensure the mesh is conforming, [1]. This is illustrated in Figure 2 in which a level 2 refinement is applied to the central triangle where dashed lines represent the bisecting edges of green triangles.

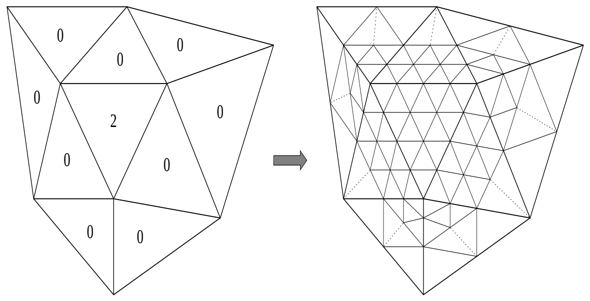


FIGURE 2. Regular and Green Refinement

De-refinement is a reversal of the refinement process, that is, the four children created through regular subdivision can be deleted, leaving the parent. Only one level of de-refinement is allowed at any one remeshing time and, in addition, all four children must be marked for deletion. De-refinement will not be allowed if a triangle in the initial mesh, produced by the mesh generator, is specified. The triangles created as a result of application of the green rule may be of poor quality and so are removed before any further mesh refinement takes place.

## *3.1 Time Integration*

Although in many time dependent PDE codes a CFL stability condition is used to control the timestep, the SPRINT2D Theta or Backward Differentiation Formula codes with functional, Newton Krylov or operator splitting methods allow automatic control of the local error. Efficient time integration requires that the spatial and temporal errors are roughly the same order of magnitude. The need for spatial error estimates unpolluted by temporal error requires that the spatial error is the larger of the two errors. The SPRINT2D software also has an option to use the strategy of Berzins, see [4], which controls the local time error to be a fraction of the growth in the spatial discretization error over a timestep.

### *3.2 Visualisation*

The driver program also allows the user to extract information about the numerical solution each time it changes or is updated. This is achieved by the user providing a *monitor* routine which SPRINT2D calls at regular intervals with a large amount of solution information. For example, each triangle has a solution value, a spatial error value and, for time-dependent problems, a temporal error value. The code can also provide a large quantity of spatial information about the unstructured mesh such as areas of triangles, lengths of edges, unit normals to edges etc. This information is used by the visualisation package which complements the SPRINT2D solver. This visualisation package is developed in IRIS GL and runs on a local host whilst SPRINT2D runs on a computationally intensive platform elsewhere. Solution frames are sent across the network to the visualisation package within which the user can interrogate the solution whilst the next frame is being calculated. An example of this is the frame shown in Figure 5 for the knock problem described above. The visualisation package displays the solution values for each triangle in the spatial mesh or error estimates in space and time. This information is not used to steer the calculation directly, but has proved to be invaluable for users learning how to apply adaptivity to their applications.

In displaying the numerical solution values for convection-dominated problems great care must be taken to avoid introducing physically unreal values not already present in the numerical solution. For example physical values of density should always be positive. The solution to convection-dominated PDE problems may have shocks and discontinuities present. Numerical PDE solvers take great care to preserve, say, the positivity of the solution. However, such discontinuities may lead to numerical undershoot and overshoot if standard interpolation techniques are used. This can mislead the user. In [11] a triangular based interpolant is described which achieves the desired properties by bounding the values it produces to be between the maximum and minimum values used to define it. This interpolant provides a more reliable and natural way for the user to view the solution.

## 4 A PSE for SPRINT2D

In designing a visually based PSE, the need for portability makes it desirable to use tools that are either industry standard or are as de-facto standards. Such tools are the X Window System and associated Widget sets which are high level X toolkits. A widget is defined as an X window with associated manipulation procedures for the window and data structures. One such set is the The Open Software Foundation (OSF) Motif Widget Set which has a distinctive look and style, and was used to construct the X interfaces for the PSE described here. It is also necessary to convert the user's information into C functions; this is done by using the Maple system which was chosen because of its wide availability, robustness, and because it provides C and TEX output.

### *4.1 A Visual Domain Specification Tool*

The Visual Domain Specification (VDS) tool aids the key task of specifying the initial domain so that it can be meshed, thus reducing the time spent on the problem specification process. The tool must provide a convenient way for the user to specify and manipulate the geometry and allow for the user to visualise a coarse mesh defined over the domain.

The VDS tool uses an internal data structure to construct the geometry. This information is then transferred to mesh generation software via a postprocessing routine. This intermediate step allows the separation of the visual specification process and the creation of the numerical domain specification file, although these two processes are closely linked.

Initially an interface was constructed to the fully automatic KSLA mesh generator [5], as used in SPRINT2D. In addition to this, an option to construct output files for the GEOMPACK and PLTMG mesh generators [7] [2] was added to demonstrate the generic nature of the tool. Overall the VDS tool is split into three main components: a drawing canvas where the user can specify the geometry using the mouse; a display canvas that shows the mesh generated from the user specified geometry; and a control panel containing buttons and labels which allows the user to control the tool.

As well as the visual components of the tool a suitable internal data structure needs to be defined. The problem faced is to design a data structure that fits around the requirements for final output format and allows flexibility for the input requirements. The description of the geometry adopted is hierarchical in that each level is built up from the lower levels, composed of vertices, lines or arcs, and regions respectively. The ability of the data structure to store geometries in a tree structure, where more complex elements are constructed from simpler elements, is one which has proved successful as it allows top-down and bottom-up manipulation of the information stored in the data structure. The structure made it possible to provide geometry output in a form suitable for GEOMPACK.

### *4.2 A Visual Problem Specification System*

The creation of the SPRINT2D driver program can be a lengthy process. The user must first define the PDE problem and the spatial domain. The

SPRINT2D modules to be used must be initialised by calling the appropriate command. The VPS system aims to provide an easy and natural way for the user to visually specify the information required by SPRINT2D. The aim is to decrease the time taken to create a valid driver program, guide the user to provide all the information needed by SPRINT2D and to avoid the need for explicit programming wherever possible.

The information needed to specify the problem may be split into: the mesh information (including boundary and initial conditions); the equation specification and finally the error control information. The VPS system uses this information to create a suitable driver program for the numerical software via the postprocessing step. At the end of each stage of this user specification process, the information supplied is stored so that the postprocessing subsystem can use it to create the driver program. For continuity, this information is used as the new default values for the user interfaces when next invoked. Although the time saved by this visual interface is important, the interface also ensures that all information is input, that sensible defaults are chosen, and that user errors are reduced.

The user defines the mesh in two stages; first, information concerning the number of boundary conditions and the mesh file is supplied. An outline of the boundary is then displayed and the boundary edges are named with unique integers. The second stage of the interface extracts and reproduces the geometry to allow the specification of the boundary conditions. The boundary conditions (Neumann, Dirichlet or Flux) are specified using Maple and mapped onto edges by placing conditions on consecutive lists of edges. The PDE functions in equation (1) and the initial conditions are also specified using Maple syntax which is then converted into the functions required by the driver program. Figure 3 shows these functions for the case study of Section 5.1. The adaptivity interface allows the user to select values for the maximum number of triangles, and to set the adaptivity and visual states to either on or off. If the adaptivity is set to on the user is prompted for the absolute and relative spatial tolerance values. Finally the user is asked to specify the integrator module (BDF or Theta) and the linear algebra module to be used (sparse, iterative or operator splitting).

## *4.3 Construction of the Driver Program*

The information provided by the users and encoded by the interfaces is then passed to the postprocessing subsystem responsible for creating the SPRINT2D driver program. The approach used is that of fitting information into a template program. The postprocessing subsystem first provides a visual summary of the information so that the user may easily validate the problem definition and then trigger the creation of the driver program. The final output of the postprocessing subsystem is a valid C driver program which can be compiled and linked to run the numerical software.

The driver program starts by including the relevant header files for the solution modules used. The initial conditions, the boundary conditions and the appropriate functions required for the finite volume method are then defined. The user may provide a monitor routine which provides a means of examining the numerical solution. This is then followed by a set of routines to instruct the numerical code where to find the previously defined functions it requires and which software packages to use. The driver program then starts the solution process by calling SPRINT2D.

The driver program, as well as having the ability to execute the numerical software, must also be easy to understand, well-structured and well-documented to allow possible user modifications. One example where this is important is the Riemann solver function required by the finite volume method. The default solver supplied by the system uses the average of the 'left' and 'right' values in the flux calculation, see Section 2.3. The user must specify the approximate Riemann solver if flow directionality has to be taken into account, see Section 5.1 for example.

## 5 Case Studies

This section will demonstrate the use of the tools by two case studies involving time dependent PDEs; other problems are in Pratt [10].

### 5.1 *Convection-Dominated PDE – Burgers' Equation*

The first problem is a Burgers' equation given by

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) + \frac{\partial}{\partial y}\left(\frac{u^2}{2}\right) = p\frac{\partial^2 u}{\partial x^2} + p\frac{\partial^2 u}{\partial y^2}$$

where $p$ is a constant defined as 0.01. The Dirichlet boundary conditions and the initial conditions on the square domain are $u(x,t) = (1+e^{\frac{x+y-t}{p}})^{-1}$ this time dependent problem is solved with the Theta integration module and iterative linear algebra. The start time is 0.15 with 15 output points and a step between output points of 0.10. Spatial adaptivity is used with absolute and relative spatial tolerance of 0.05 and a maximum number of 10,000 triangles. This example demonstrates the construction of the approximate Riemann solver by the postprocessing subsystem for the driver program. The routine from the driver program for the averaging Riemann solver is given below; its use results in negative solution values close to the wave front.

```
void problem_rs(TRIAD_Line *line, int npde, double x,
            double y, double t, int sub_name,
            double norm_x, double norm_y, double u_l[],
            double u_r[], void *users_data, double nf[])
```

```
{ /*  Burgers eqn: crude averaging Riemann solver */
  double u = ( u_l[0] + u_r[0] ) / 2.0 ;
  double f_x, f_y;
  f_x = 0.5*u*u;
  f_y = 0.5*u*u;
  nf[0] = f_x * norm_x + f_y * norm_y;
} /* Riemann solver */
```

The negative values vanish when Roe's Riemann solver, see [9] is implemented by inserting the code:

```
  /* Burgers eqn: Roe fix to averaging Riemann solver */
  if u > 0.0 u = u_l[0];
  else       u = u_r[0];
```

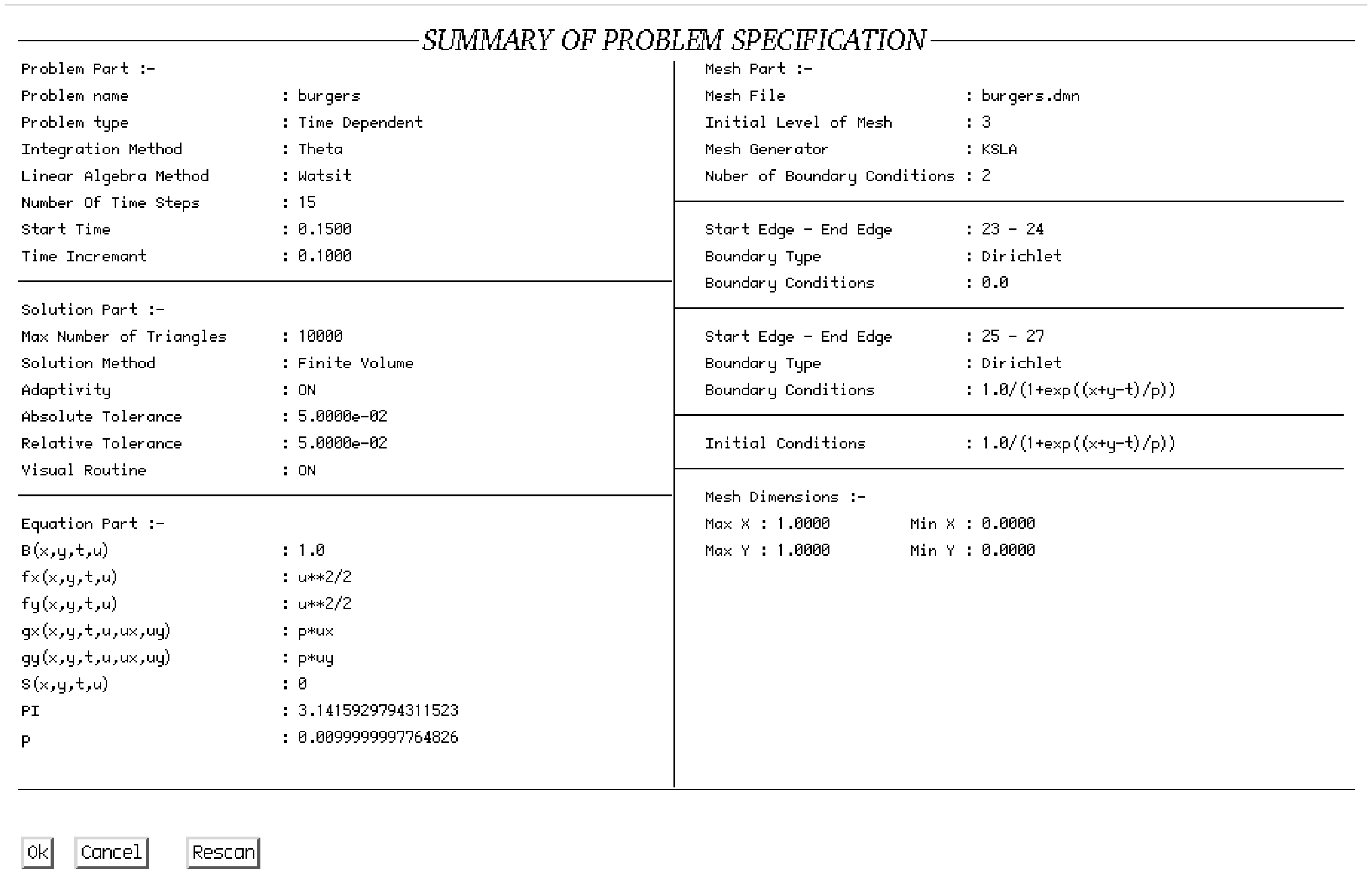
before the assignment to `f_x` . This example shows that although the VPS tool does not know how to produce a correct Riemann solver for a general problem, the code it does produce may be easily modified by the user.

## *5.2 Combustion Knock-Modelling Problem*

A challeng test problem is a combustion model relating to the modelling of 'knock' in car engines, see [8]. The model is used to investigate the effects of autoignition in end gases in an idealised car engine cylinder. The onset of 'knock' is seen when large pressure pulses interact with the edges of the cylinder. Mathematically the problem is specified by a system of five PDEs representing conservation of mass, momentum and energy together with a species equation. The functions in equation (1.1) are defined by: $\underline{U} = (\rho, \rho u, \rho v, E, \rho z)^T$, and

$$\underline{f^x} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u[E+p] \\ \rho uz \end{pmatrix}, \underline{f^y} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v[E+p] \\ \rho vz \end{pmatrix}, \underline{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -\rho z \; k(T) \end{pmatrix}$$

and $k(T) = exp(\hat{\beta}\left(1 - \frac{1}{T}\right))$ , $T = P/\rho$ and $\hat{\beta} = 20.0$. The variables $\rho, u, v, p$ are the density, the velocities in the x and y dimensions and the pressure respectively. The variable $z$ represents the scaled fuel concentration. The energy $E$ is defined by the equation of state $E = \frac{p}{(\gamma-1)} + \frac{\rho u^2 + \rho v^2}{2} + \alpha\rho z$ where $\gamma = 1.2$ and $\alpha = 8.0$ . The geometry of the problem is shown in Figure 4 in which the irregular solid line represents the initial position of the flame front, as taken from experimental data. The area to the left of this front contains unburnt fuel while that to the right is one in which the fuel has burnt. The dotted concentric circles indicate temperature hot



```
SUMMARY OF PROBLEM SPECIFICATION

Problem Part :-                                      Mesh Part :-
Problem name               : burgers                 Mesh File                    : burgers.dmn
Problem type               : Time Dependent          Initial Level of Mesh        : 3
Integration Method         : Theta                   Mesh Generator               : KSLA
Linear Algebra Method      : Watsit                  Nuber of Boundary Conditions : 2
Number Of Time Steps       : 15
Start Time                 : 0.1500                  Start Edge - End Edge        : 23 - 24
Time Incremant             : 0.1000                  Boundary Type                : Dirichlet
                                                     Boundary Conditions          : 0.0
Solution Part :-
Max Number of Triangles    : 10000                   Start Edge - End Edge        : 25 - 27
Solution Method            : Finite Volume           Boundary Type                : Dirichlet
Adaptivity                 : ON                      Boundary Conditions          : 1.0/(1+exp((x+y-t)/p))
Absolute Tolerance         : 5.0000e-02
Relative Tolerance         : 5.0000e-02              Initial Conditions           : 1.0/(1+exp((x+y-t)/p))
Visual Routine             : ON
                                                     Mesh Dimensions :-
Equation Part :-                                     Max X : 1.0000        Min X : 0.0000
B(x,y,t,u)                 : 1.0                     Max Y : 1.0000        Min Y : 0.0000
fx(x,y,t,u)                : u**2/2
fy(x,y,t,u)                : u**2/2
gx(x,y,t,u,ux,uy)          : p*ux
gy(x,y,t,u,ux,uy)          : p*uy
S(x,y,t,u)                 : 0
PI                         : 3.1415929794311523
P                          : 0.0099999999764826
```

FIGURE 3. Summary Window for Burgers' Problem

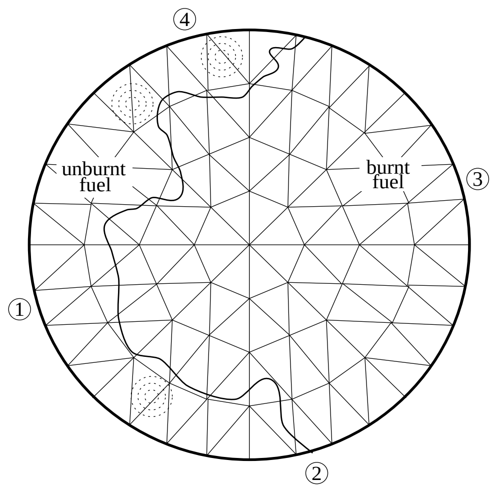


FIGURE 4. Diagram of Knock Model.

spots which will lead to autoignition and pressure pulses travelling across the cylinder to cause 'knock'. Points numbered 1 to 4 are the four pressure transducers at which experimental time histories of pressure are available.

The initial conditions are as follows. The initial velocities $u$ and $v$ are zero; the pressure has the value $p = 1$; the fuel concentration $z$ is zero in the burnt region and one in the unburnt region. The scaled temperature $t$ is 0.75 in the unburnt region except at the hot spots where it rises to one and in the burnt region it has value $1 + \alpha(\gamma - 1)/\gamma$, $\rho = p/T$ . The quantities $E, \gamma$ and $\alpha$ are defined by above equation of state. The implementation of the reflective boundary conditions is described in Section 2.2.

Although fixed mesh solutions to this problem are given in [4], the focus here will be on the effect of adaptivity and the PSE on the solution process. The problem is non-standard in that the initial conditions cannot be specified by a mathematical function but are specified on a triangle by triangle basis from camera data and read in from a data file. Soon after integration starts the complex flow patterns for this problem mean that heavy mesh refinement occurs. Thus it is useful to let the user refine the mesh a priori. To allow this the adaptivity module was modified to allow user specified mesh refinement around a specific location.

The SPRINT2D code was employed with the Theta and operator splitting options as in [4]. Runs were performed with fixed triangular meshes with 2048 and 8192 elements respectively and adaptive meshing with the maximum number of triangles set to 8192 and 32768 respectively. These modes are referred to in the results table as FIXED and ADAPT respectively. The SPRINT2D code was used with standard local error control with absolute tolerances of $10^{-4}$ for all the PDE variables except the species concentration $z$ for which $10^{-5}$ is used. A maximum stepsize of $5.0 \times 10^{-4}$ was

TABLE 1.1. Transducer 1 Pressure Spike

| Code | *Mesh* | MODE | TIME | PEAK |
|---|---|---|---|---|
| LUMAD | 10000 | CFL | 32.08 | 5.87 |
| SPRINT2D | 2048 | FIXED | 41.09 | 4.44 |
| SPRINT2D | 8192 | FIXED | 28.59 | 5.61 |
| SPRINT2D | 8192 | ADAPT | 28.63 | 5.59 |
| SPRINT2D | 32768 | ADAPT | 27.12 | 6.45 |

imposed during the initial combustion phase in order to prevent unphysical solution values being passed into the Riemann solver. The geometry of the problem and the solution shortly after the start of integration are shown in Figure 5, which shows the output the user sees from SPRINT2D.

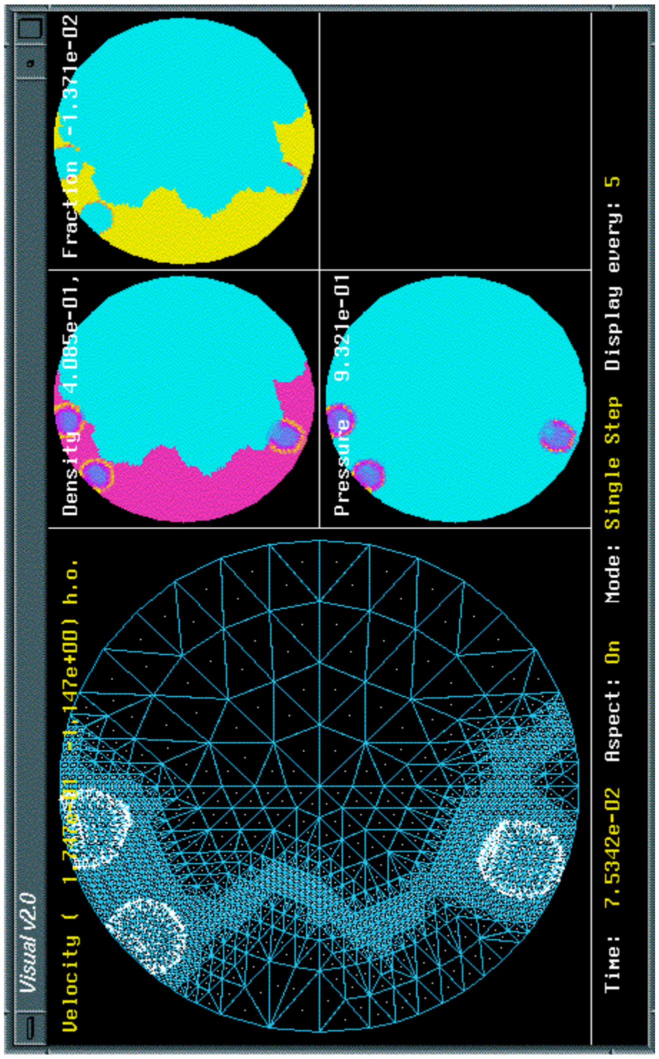
In contrast, the fixed timestep regular square mesh code LUMAD, [4], uses an ad-hoc Riemann solver approach to determine the flux values. Timestepping is done using the forward Euler method with only a CFL condition to control the timestep. In order to obtain results consistent with SPRINT2D and to resolve the reaction transients, LUMAD must use a square regular mesh with 100x100 mesh points and 40,000 timesteps giving a CFL number of 0.01.

The entries marked TIME show the time of the peak pressure pulse at pressure transducer 1. PEAK indicates the values of this peak. The physical significance of the PEAK value is that it indicates the strength of the pressure pulse that causes 'knock' while the TIME value indicates when this occurs. Correct computation of these values is thus important if the mathematical model is to be validated against experiments.

The results in Table 1 show a consistent trend with those of [4]. Moreover by the time the adaptive run has encountered the maximum pressure spike the mesh has about 24,000 triangles, with the finest mesh in the region of the pressure spikes. In this case the spatial refinement tolerances are $10^{-4}$ for density and $10^{-2}$ for the other PDE variables and hence refinement is biased towards the density errors. It is worth stating that a poor choice of tolerances can lead to inappropriate refinement.

One of the challenging aspects of this problem is that of writing a good physically realistic Riemann solver [4]. For this problem the Riemann solver function within the driver program is approximately 650 lines of code and comments and took an expert user of SPRINT2D about 5 days to write and debug.

The table shows that unless great care is taken with the choice of time step and spatial mesh, over-large pressure pulses at incorrect times may be recorded. In particular the use of adaptivity enables the mesh to be concentrated where it is needed.



```
Velocity ( 1.747e-01 -1.147e+00) h.o.
Density 4.085e-01, Fraction -1.371e-02
Pressure 9.321e-01
Time: 7.5342e-02  Aspect: On  Mode: Single Step  Display every: 5
```

FIGURE 5. SPRINT2D Solving the Knock Problem

## 6 Conclusions

The aim in this paper has been to show how SPRINT2D and its associated PSE together from a powerful and semi-automatic way of solving time-dependent PDEs. The software fits into the PSE framework described in Section 1 by using Maple to describe the PDEs, the VDS tool to model the geometry and GEOMPACK to generate the mesh. The SPRINT2D finite volume scheme and associated error indicators are used to compute the solution and adapt the mesh, with the visual module and IRIS Explorer being used to display the results. Although not described here a prototype distributed parallel version of the code exists, [15]. The overall approach has proved successful for a broad range of convection-dominated problems with complex geometries needing adaptivity. The modularity of the software does make it possible to devise efficient components for particularly important and/or difficult problems; one example of this being the operator splitting iterative scheme used in [4].

The combination of end-users and the developers of SPRINT2D and its PSE has helped to construct a package with numerical reliability, has eased the solution process, reduced the time spent and provided a more natural and convenient way to solve the PDEs. The users were enthusiastic about the VDS and VPS tools and about how easy it was to generate working code. The benefit of this however must be balanced against the many months of effort spent on the knock problem experimenting with different meshes, tolerances, Riemann solvers and initial conditions.

Perhaps the largest problem faced with automatically producing a driver program capable of giving a valid solution is that of the Riemann solver. In the case of problems with source terms the best Riemann solver may not even be known. However, as understanding in this area increases, PSEs can develop alongside the numerical code.

This work has shown that it is possible to utilise current scientific computing technology to build software tools and packages that when combined form an easy-to-use layer surrounding complex computational code. This layer can help both novice and experienced users to better utilise their time, efforts and knowledge, even if the layer provides only partial help for difficult problems and specialised modules may still need to be written.

**Acknowledgements**
Thanks are due to L.E. Scales of Shell Research and J.G.Pan and C.G.W. Sheppard of the Mechanical Engineering Dept. at Leeds. This research was supported by EPSRC and Shell Research UK.

## 7 References

[1] S Adjerid, J E Flaherty, P K Moore, and Y J Wang. High-order adaptive methods for parabolic equations. In J M Hyman, editor, *Experimental Mathematics: Computational Issues in Non-Linear Science*,

Physics D 60 1-4, pages 94–111. North-Holland, 1992.

[2] R E Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations Users' Guide 7.0*, SIAM, Philadelphia,1995.

[3] M Berzins, P H Gaskell, A Sleigh, W Speares, A Tomlin, and J M Ware. An adaptive CFD solver for time dependent environmental flow problems. pp.311-318 in Numerical Methods for Fluid Dynamics V, Eds K.W.Morton and M.J.Baines, Clarendon Press, Oxford 1995.

[4] M Berzins and J M Ware. Solving convection and convection reaction problems using the M.O.L. *Appl. Num. Math.* , 20:83–99, 1996.

[5] R M Furzeland, P C Rem, and R F Van der Wijngaart. General purpose software for multi-dimensional partial differential equations. Tech. Rep., Shell Research Amsterdam, 1989.

[6] E Gallopoulos, R Houstis, and Rice J R. Future directions in problem solving environments for computational science. Technical report, NSF Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry and Artificial Intelligence for Computational Science, Washington, D.C., April 1991.

[7] B Joe. GEOMPACK - a software package for the generation of meshes using geometric algorithms. *Adv. Eng. Soft.*, 13(5/6):325–331, 1991.

[8] J. Pan and C.G.W. Sheppard, A theoretical and experimental study of the modes of end gas autoignition leading to knock in an SI engine. *S.A.E. paper* 94-2060 (1994). S.A.E., Warrendale, PA 15096, USA

[9] S V Pennington and M Berzins. New NAG library software for first-order partial differential equations. *ACM Transactions on Mathematical Software*, 20(1):63–99, March 1994.

[10] P R Pratt. Problem Solving Environments for the Numerical Solution of P.D.E.s Ph.D. Thesis, University of Leeds, 1995.

[11] P R Pratt and M Berzins. Shock Preserving Quadratic Interpolation for Visualisation on Triangular Meshes. *Comput. and Graphics*, 20(5) 1996.

[12] L E Scales. NAESOL: User's guide. Internal report, Shell Research Ltd, Chester, 1993.

[13] H J Stetter. Tools for scientific computation. *Zeitschrift für Angewandte Mathematik und Mechanik ZAMM*, 73(12):335–348, 1993.

[14] J M Ware. *The Adaptive Solution of Time-Dependent Partial Differential Equations in Two Space Dimensions.* PhD thesis, School of Computer Studies University of Leeds, 1993.

[15] C M Walshaw and M Berzins. Dynamic load balancing for PDE solvers on adaptive unstructured meshes. *Concurrency Practice and Experience* 7:7-28, 1995.