# Sarande
# A Numerical Package for Solving Reaction-Diffusion Equations on Arbitrary Surfaces

Stephan Wehner

August 6, 1999

## Abstract

Sarande is a package for solving Reaction-Diffusion equations numerically on arbitrary surfaces. The form of the Reaction-Diffusion equations that can be specified is

$$\begin{aligned}\frac{\partial X}{\partial t} &= D_X \Delta^2 X + f_X(X,Y)\\ \frac{\partial Y}{\partial t} &= D_Y \Delta^2 Y + f_Y(X,Y) \qquad (1)\end{aligned}$$

where $X, Y$ are the two functions defined on the curved domain and whose evolution over time is sought. $\Delta^2$ denotes the Laplacian operator, $D_X, D_Y$ are constants, and $f_X, f_Y$ are arbitrary functions. Boundary conditions may be chosen to be Dirichlet or von Neumann: $\frac{\partial X}{\partial \mathbf{n}} = \frac{\partial Y}{\partial \mathbf{n}} = 0$ on the boundary.

The surface is specified through a triangulation: a set of nodes with three dimensional coordinates and a list of neighbours for each node, sorted consistently either clockwise or counterclockwise.

The package is written in the programming language C. It includes programs to generate triangulations of the disk.

Test cases are described to illustrate the range of correctness of the package.

## 1 Outline

The aim of this document is to orient the reader about the package and to enable them to use it. It will provide a simple introduction to the particular numerical methods employed.

We will first describe the mathematical manipulations to the system which lead to a linear system. Then we will describe how to install the package and run it on the included test cases.

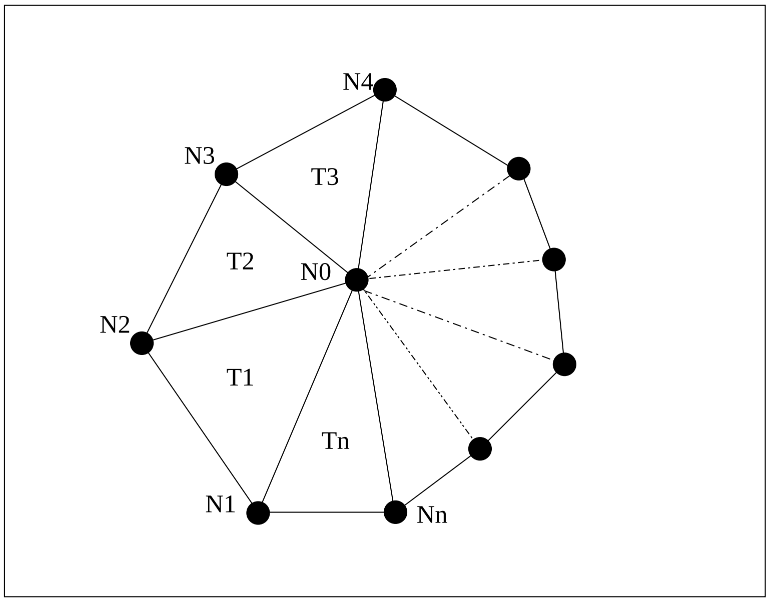

Figure 1: Arbitrary Element with labelled nodes

## 2 Method

The Finite Element Method is employed in order to extract at every timestep a system of ordinary differential equations. This system in turn approximates at every node the evolution of the solution over the timestep. It is solved through a Runge-Kutta method following the suggestion of Steve Ruuth and others [1] requiring the calculation of solutions to sparse systems of linear equations. These are approximated by a preconditioned Bi-Conjugate Gradient solver, as described in [2].

### 2.1 Finite Element Method

The approach of the Finite Element Method is to restrict attention to a finite subset of the domain, called *element*. One seeks to describe the evolution of the function at a specified point within that domain. The equation is multiplied with a well-chosen function and then integrated over the given element (which in our case is an area), in order to extract an equation describing the evolution of the solution at a given node, as it depends on the values at its neighbours.

The details are as follows. Figure 2.1 shows an arbitrary element with one center node $N_0$ and $n$ nodes along the boundary of the element labelled $N_1, \ldots, N_n$. There are also triangles labelled $T_1, \ldots, T_n$. The union of the triangles will be denoted by $\Omega$, the boundary will be denoted by $\Gamma$.

To start with the nodes are placed in 3-dimensional space. However, we will

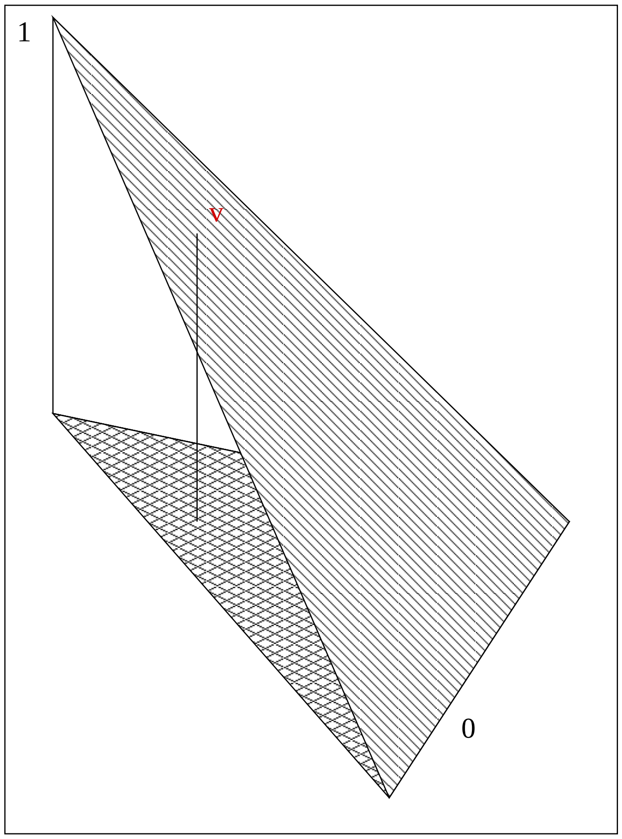

Figure 2: Function $v$ over one triangle

assume that the triangulation approximates the curved surface well so that we can treat the element as flat: we can find a plane so that the projection of the element onto this plane describes the behaviour of the solution at the nodes of the element.

We treat the two PDE's separately and look at an equation of the form

$$\frac{\partial U}{\partial t} = \Delta^2 U + f(U) \tag{2}$$

neglecting the diffusivity-factors $D_X, D_Y$ for the time being. Let $v$ be a function which has the value 1 at $N_0$, is 0 on $\Gamma$ and linear otherwise: see Figure 2.1.

Multiply equation (2) by $v$ and integrate over $\Omega$ to obtain

$$\int_\Omega v\frac{\partial U}{\partial t} = \int_\Omega v\Delta^2 U + \int_\Omega v f(U) \tag{3}$$

We will approximate both sides and derive an ordinary differential equation relating the time-dependent variables $U(N_0), U(N_1), \ldots, U(N_n)$. We will write $U_0, \ldots, U_n$.

### 2.1.1 Approximation of left hand side of equation (3)

We approximate $\int_\Omega v \frac{\partial U}{\partial t}$ as a product according to

$$\begin{aligned} \int_\Omega av &= \sum_{T_i} \int_{T_i} av \\ &\approx \sum_{T_i} \{a \text{ at centre}\}\{v \text{ at centre}\} \text{ Area}(T_i) \\ &\approx \sum_{T_i} \frac{a_{N0} + a_{N_l^i} + a_{N_r^i}}{3} \frac{1}{3} \text{ Area}(T_i) \end{aligned} \tag{4}$$

Here we write $N_l^i$ and $N_r^i$ to refer to the nodes of triangle $T_i$ which are on the boundary so that $N_l^i$ occurs before $N_r^i$ in a clockwise ordering of the node. So we have for the left hand side of equation (3) the approximation

$$\int_\Omega v \frac{\partial U}{\partial t} \approx \sum_{T_i} \frac{\frac{\partial U_0}{\partial t} + \frac{\partial U_l^i}{\partial t} + \frac{\partial U_r^i}{\partial t}}{3} \frac{\text{Area}(T_i)}{3} \tag{5}$$

Again, $U_l^i$ and $U_r^i$ refer to the value of U at the boundary nodes of triangle $T_i$.

### 2.1.2 Approximation of right hand side of equation (3)

The integration of the $vf$ term follows the above, yielding

$$\int_\Omega vf \approx \sum_{T_i} \frac{f_0 + f_l^i + f_r^i}{3} \frac{\text{Area}(T_i)}{3} \tag{6}$$

According to Green's identity, we have

$$\int_\Omega v\Delta^2 U = \oint_\Gamma v \cdot \frac{\partial U}{\partial n} - \int_\Omega \nabla U \cdot \nabla v \tag{7}$$

Note that at interior elements we have $v = 0$ on the boundary $\Gamma$. For Dirichlet boundary conditions, an equation for the values on the boundary is not required: the value at the next time step is already known. In the case of von Neumann boundary conditions, we have $\frac{\partial U}{\partial n} = 0$ on the boundary. Regardless of the boundary conditions the term integrating over the boundary of $\Omega$ does not come into play.

We have

$$\int_\Omega v\Delta^2 U = -\int_\Omega \nabla U \cdot \nabla v \tag{8}$$

$$= \quad -\sum_{T_i}\int_{T_i}\nabla U\cdot\nabla v \tag{9}$$

$$\approx \quad -\sum_{T_i}(\nabla U\cdot\nabla v)_i\mathrm{Area}(T_i) \tag{10}$$

assuming $\nabla U$ and $\nabla v$ are constant on one triangle, the computation of which will be dealt with next:

For a function $z : R^2 \to R$ parametrizing a plane in $R^3$ such that

$$z(x_l, y_l) = z_l, z(x_r, y_r) = z_r, z(x_0, y_0) = z_0$$

we have the following

$$\begin{pmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \end{pmatrix} = \frac{-1}{2A}\begin{pmatrix} z_l(y_r - y_0) + z_r(y_0 - y_l) + z_0(y_l - y_r) \\ z_l(x_0 - x_r) + z_r(x_l - x_0) + z_0(x_r - x_l) \end{pmatrix} \tag{11}$$

where $A$ denotes to the area of the triangle formed by the three points.

Therefore, $\nabla v$ for $v_0 = 1, v_l = v_r = 0$ is

$$\begin{pmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{pmatrix} = \frac{-1}{2A}\begin{pmatrix} y_l - y_r \\ x_r - x_l \end{pmatrix} \tag{12}$$

when we choose $z_0 = 1, z_r = z_l = 0$ and $x_0 = y_0 = 0$.

Similarly, we obtain for $U$ defined on a triangle $T$ with nodes $N_0 = (0,0)$, $N_l = (x_l, y_l)$ and $N_r = (x_r, y_r)$

$$\begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix} = \frac{-1}{2A}\begin{pmatrix} U_0(y_l - y_r) + U_l y_r - U_r y_l \\ U_0(x_r - x_l) - U_l x_r + U_r x_l \end{pmatrix} \tag{13}$$

so that in fact the first term of the right-hand side of equation (3) is

$$\sum_{T_i}\frac{1}{4\mathrm{Area}(T_i)}\begin{pmatrix} y_l^i - y_r^i \\ x_r^i - x_l^i \end{pmatrix}\begin{pmatrix} U_0(y_l^i - y_r^i) + U_l^i y_r^i - U_r^i y_l^i \\ U_0(x_r^i - x_l^i) - U_l^i x_r^i + U_r^i x_l^i \end{pmatrix} \tag{14}$$

where we are taking the scalar product of two vectors.

The ODE obtained is thus

$$\begin{aligned}\textstyle\sum_{T_i}\frac{1}{9}\mathrm{Area}(T_i)\left(\frac{\partial U_0}{\partial t} + \frac{\partial U_l^i}{\partial t} + \frac{\partial U_r^i}{\partial t}\right) = \\ \textstyle\sum_{T_i}\frac{1}{4\mathrm{Area}(T_i))}[U_0\{(y_l - y_r)^2 + (x_r - x_l)^2\} + \\ U_l\{(y_l - y_r)y_r - (x_r - x_l)x_r)\} \\ U_r\{-(y_l - y_r)y_l + (x_r - x_l)x_l\}] \\ +\tfrac{1}{9}\mathrm{Area}(T_i)(f(U_0) + f(U_l) + f(U_r))\end{aligned}$$

from which one can read of the coefficents of the functions $U_0, \ldots, U_n$.

## 2.2 Summary

If one numbers the nodes from 1 to $N$, one may talk about vectors $\mathbf{X}$ and $\mathbf{Y}$ - the $i$-th entry of for example $\mathbf{X}$ refers to the value of $X$ at the node with number $i$. Then one may formulate the above approximations in terms of matrices. Equation (1), which is a partial differential equation, has been translated into a system of ordinary differential equations. Its form is

$$\begin{aligned} M_A \mathbf{X}_t &= D_X M_L \mathbf{X} + M_A f_X(\mathbf{X},\mathbf{Y}) \\ M_A \mathbf{Y}_t &= D_Y M_L \mathbf{Y} + M_A f_Y(\mathbf{X},\mathbf{Y}) \end{aligned} \tag{15}$$

Subscript $t$ denotes differentiation with respect to time. Note that we have added the diffusivities $D_X, D_Y$, which we had dropped in the discussion above. The matrix $M_A$ corresponds to the coefficients calculated in the right hand side of equation (5). Matrix $M_L$ corresponds to the coefficients in the right hand side of equation (14). Note that these matrices are sparse: there is an entry at row $i$ and column $j$ if and only if the nodes numbered $i, j$ lie on a triangle. Please convince yourself that while $M_A$ is symmetric, $M_L$ is not.

The original problem has thus been transformed into a form to which the method described in [1] can be applied – up to the solution of linear systems which are produced by that method.

# 3 Solution of the System of ODE's

The system (15) derived in the previous section may be solved with a Runge-Kutta method, following the suggestion in [1]. First let us rewrite the system (15) in a single equation:

$$\begin{pmatrix} M_A & 0 \\ 0 & M_A \end{pmatrix} \mathbf{U}_t = \begin{pmatrix} D_X M_L & 0 \\ 0 & D_Y M_L \end{pmatrix} \mathbf{U} + \begin{pmatrix} M_A & 0 \\ 0 & M_A \end{pmatrix} \begin{pmatrix} f_X(\mathbf{X},\mathbf{Y}) \\ f_Y(\mathbf{X},\mathbf{Y}) \end{pmatrix} \tag{16}$$

where

$$\mathbf{U} = \begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix}, \mathbf{U}_t = \begin{pmatrix} \mathbf{X}_t \\ \mathbf{Y}_t \end{pmatrix}$$

Let us write

$$\mathcal{A} := \begin{pmatrix} M_A & 0 \\ 0 & M_A \end{pmatrix}$$

and

$$\mathcal{L} := \begin{pmatrix} D_X M_L & 0 \\ 0 & D_Y M_L \end{pmatrix}$$

In Section 2 of [1] various schemes are given in terms of numbers $a_{i,j}$ and $\hat{a}_{i,j}$. We apply these to the system at hand here in the following way. Their $f$ is our $\mathcal{A}^{-1}\mathcal{L}$ and their $g$ is our

$$\begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix} \mapsto \begin{pmatrix} f_X(\mathbf{X},\mathbf{Y}) \\ f_Y(\mathbf{X},\mathbf{Y}) \end{pmatrix}$$

We have $\mathbf{U}_{n-1}$ in terms of the vectors $\mathbf{X}_{n-1}, \mathbf{Y}_{n-1}$ at time $(n-1)\Delta t$ and will determine an approximation $\mathbf{U}_n$ at time $n\Delta t$. The steps in [1], adapted to the present situation, read as follows:

---

1. Set
$$\hat{\mathbf{K}}_1 := \begin{pmatrix} f_X(\mathbf{X}_{n-1}, \mathbf{Y}_{n-1}) \\ f_Y(\mathbf{X}_{n-1}, \mathbf{Y}_{n-1}) \end{pmatrix}$$
2. For $i = 1, \ldots, s$, do:
   (a) Solve $\mathcal{A}\mathbf{K}_i = \mathcal{L}\mathbf{U}_i$, where
   $$\mathbf{U}_i = \mathbf{U}_{n-1} + \Delta t \sum_{j=1}^{i} a_{i,j}\mathbf{K}_j + \Delta t \sum_{j=1}^{i} \hat{a}_{i+1,j}\mathbf{H}_j$$
   (b) Set $\hat{\mathbf{K}}_{i+1} := \begin{pmatrix} f_X(\mathbf{X}_i, \mathbf{Y}_i) \\ f_Y(\mathbf{X}_i, \mathbf{Y}_i) \end{pmatrix}$
3. Set
$$\mathbf{U}_n := \mathbf{U}_{n-1} + \Delta t \sum_{j=1}^{s} b_j\mathbf{K}_j + \Delta t \sum_{j=1}^{\sigma} \hat{b}_j\hat{K}_j$$

---

Note that 2a. asks to solve linear systems of the form

$$(\mathcal{A} - a_{i,i}\mathcal{L})\mathbf{K}_i = \mathcal{L}(\mathbf{U}_{n-1} + \Delta t \sum_{j=1}^{i-1} a_{i,j}\mathbf{K}_j + \Delta t \sum_{j=1}^{i} \hat{a}_{i+1,j}\mathbf{H}_j)$$

The matrix $\mathcal{A} - a_{i,i}\mathcal{L}$ will be non-symmetric in general, because $M_L$ is.

## 4 The Sarande Code

The code can be found at `http://www.math.sfu.ca/~wehner/Sarande/code.tar.gz`. With the Unix command

```
gunzip -c code.tar.gz | tar -xvf -
```

a directory `Sarande_code` is unpacked which contains

1. A directory SRD_vN, for von Neumann Boundary conditions
2. A directory SRD_Dir, for Dirichlet Boundary conditions
3. A directory disk_triangulate, for generating triangulations of the unit disk

These directories contain C-code and makefiles. Issuing in these three directories the Unix command

```
make
```

will compile the three executables

1. `testsrd_vN`
2. `testsrd_Dir`
3. `triangulate`

The `disk_triangulate` directory also contains two files `plottriangulation.c` and `table.c` which may be of some use. Typing `make plottri` will compile `plottriangulation.c` and produce an executable `plottri`. With `plottri` one may convert the output of `triangulate` to gnuplot input, so that one can see the structure of the triangulation. Typing `make table` will produce the executable `table` which allows one to see how many nodes there are in triangulations depending on their parameters.

There is a directory `test` in both directories `SRD_vN` and `SRD_Dir`, which contain two (`csh`) shell scripts `simpletest` and `runtest`. Just issuing the Unix commands

```
./simpletest
```

or

```
./runtest
```

in the `test` directories should cause the tests to run. They solve a linear Reaction-Diffusion System of the form

$$\begin{aligned} \frac{\partial X}{\partial t} &= D_X \Delta^2 X + aX + bY \\ \frac{\partial Y}{\partial t} &= D_Y \Delta^2 Y + cX + dY \end{aligned} \qquad (17)$$

on a disk (for the von-Neumann boundary conditions) and on a hemisphere (for the Dirichlet boundary conditions). The parameters are chosen so that a particular harmonic has positive growth rate. This harmonic is taken as the initial distribution. The `simpletest` script just runs for one triangulation and one value of $\Delta t$. The programs `testsrd_Dir/vN` calculate a simple $l_2$-error and write it to a file called `error`. When `simpletest` ends results should be found in a directory `R_25_1`, where the error file can be inspected. It should show an error of 0.005162 at $t = 10$ (von Neumann) and 0.0866 at $t = 10$ (Dirichlet), the error changing linearly with time.

Both `runtest` scripts might run for half a day since they solve the problem with triangulations of four different sizes and with four different small choices of $\Delta t$.

The `runtest` script also writes a file `ploterror` which can be used to plot the errors from all computations using gnuplot with the gnuplot command

```
load "ploterror"
```

It should be possible to read off from the `simpletest` and `runtest` code how to invoke the executables `testsrd_Dir` and `testsrd_vN`. The programs are set up to take their parameters from the shell's environment variables. One of these parameters is called `SRD_result_dir` and specifies a directory in which results of the computation are stored in files. The programs also produce files `HOW` in this directory With the `csh` command

```
source HOW
```

in the result directory one may set the environment variables to the values which produced the results. It is also easy to change these values with the `csh` command `setenv` (see the `csh` manual pages).

## 4.1 The C-files

It seems the easiest to list all C-files in the order in which they are executed when the programs are run:

1. `main.c` This file defines the function `main` which calls all other high level functions, defined in the following C-files.
2. `parameters.c` To read the parameters to the program, which are expected to be defined as shell environment variables.
3. `init_surface.c` To read the triangulation from a file specified as the parameter `SRD_triangulation`; also calling code from `allocation.c` to allocate memory for all arrays involved.
4. `allocation.c` Allocation of memory for all arrays using `malloc`. Also tests if memory allocation was successfull.
5. `test_distribution.c` This file defines functions to calculate the exact solution chosen and and compute errors. The errors are written to a file called `error` in the directory specified by the parameter `SRD_result_dir`.
6. `setup_matrices.c` Code to compute the coefficients in the matrices $\mathcal{A}$ and $\mathcal{L}$ from Section 3.
7. `triangle.c` Code to compute functions relating to triangles.
8. `loop.c` The main computation loop is defined in this file. There is also code to generate ouput telling how far the computation has progressed.
9. `reaction_diffusion.c` The code which implements the algorithm from Section 3. One of the schemes of the paper [1] is selected, others have been coded but are not enabled.
10. `solve_system.c` The implementation of the Bi-Conjugate Gradient Solver for the linear systems generated by the Runge-Kutta code.

11. `solve_system_DEPCY.c` Parts of the Bi-Conjugate Gradient Solver which depend on $X$ and $Y$.

12. `dump.c` Defines functions to dump the current state of the computation to a file, and to read such a state from such a file.

These files and their function are the same for the Dirichlet and the von-Neumann code. The differences are in the

1. exact solutions chosen for testing, see files `test_distribution.c`,

2. in the definition of matrix coefficients in $\mathcal{A}$ and $\mathcal{L}$ in the file `setup_matrices.c`,

3. in the Runge-Kutta code which does not change the values for the boundary nodes in the case of Dirichlet boundary conditions (see both files `reaction_diffusion.c`).

The directory `include` contains all the header files required. There is also a file `real.h` which defines a macro `REAL` to be `double`. Computations could be done with floats by defining this macro to be `float`. Then there is the file `globals.h` which defines the global variables and some macros like `ABS` and `MAX`. It also defines macros to access some two-dimensional arrays.

A file `BM.h` defines macros `IF, THEN, ELSE, BEGIN, END` ... so that the C-code can be written in PASCAL style.

## 5 Acknowledgements

Weiming Cao showed me the approach of the Finite Element Method. Chen Grief helped me with the linear system solver. Many thanks to both!

## References

[1] Uri Ascher, Steven Ruuth, Raymond Spiteri; *Implicit-Explicit Runge-Kutta Methods for Time-Dependent Partial Differential Equations*, 1997

[2] R. Barrett, M. Berry, T. F. Chan,J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst; *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, 1994, Philadelphia, PA, also `http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html`

[3] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1996, Boston, MA