1996

GAUSS: An Automatic Algorithm Selection System for Quadrature

N. Ramakrishnan

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
96-048

GAUSS: AN AUTOMATIC ALGORITHM SELECTION SYSTEM FOR QUADRATURE

Narendran Ramakrishnan
John R. Rice

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR 96-048
August 1996

GAUSS: An Automatic Algorithm Selection System for Quadrature

N. Ramakrishnan and J.R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN47906

Abstract

This paper introduces GAUSS - a system for the automatic selection of quadrature routines in numerical computation. Given a problem in numerical integration and constraints on time and accuracy, GAUSS comes up with an efficient algorithm to solve it. It banks on a performance database of various algorithms and test problems, an automatic feature identification module and a knowledge methodology that represents information about the numerical domain in terms of predicate rules. Inductive Logic Programming is used to induce learning and make inferences in this domain. Experimental Results are presented. It is found that GAUSS comes up with several heuristics that have been earlier formulated by the domain experts.

1 Introduction

In this paper, we study the task of algorithm selection [10] for problems in numerical computation. The scope of the current study is restricted to the problem of numerical quadrature. The algorithm selection problem for this domain can be formally stated as follows (one–dimensional) [4]:

Select an algorithm to evaluate

$$I = \int_a^b f(x)dx$$

so that relative error $\epsilon_r < \theta$ and N_i is minimized,

where θ is an user–specified error requirement and N_i is the number of 'function evaluations' i.e., the number of times $f(x)$ is evaluated in the region $[a, b]$ to yield the quadrature rule of the desired accuracy. ϵ_r and N_i are chosen as performance criteria because:

- For most software implementations of integration routines, an absolute error ϵ_a and a relative error ϵ_r are input. For the integral $I = \int_a^b f(x)dx$, these routines compute $\{R_{n_k}, E_{n_k}\}$ where the first term in the tuple is the estimate of the integral using n_k values of $f(x)$ while E_{n_k}

is the relevant error estimate for R_{n_k}. Following a practice set by de Boor [5], the automatic quadrature routines terminate when the error condition

$$| R_{n_k} - I | \leq E_{n_k} \leq max(\epsilon_a, \epsilon_r \mid R_{n_k} |)$$

is satisfied. In most of the literature (on performance evaluation of numerical integration software) and in a majority of the implementations, the routines are made to impose a strictly relative accuracy by setting ϵ_a to zero. Thus, we have chosen ϵ_r to be the main accuracy criterion.

- The time required to evaluate an integral by a numerical technique seems to vary quite widely from one implementation to another, even for the same generic technique (method) with the same number of 'nodes'. Moreover, most efficient quadrature routines are of the adaptive nature so that the weights and nodes are chosen dynamically during the computation. Thus, a more uniform metric seems to be the number of function evaluations (N_i) required to determine an integral. From the experiments conducted, these seem to be fairly constant over the wide range of implementations available. Also, a very popular integration package, QUADPACK [9] uses this metric to analyze the efficiency of its algorithms.

- These criteria were selected with a view toward keeping the initial software simple. Further performance measures can be included once the basic problems of algorithm selection in this domain have been understood. The program developed for this purpose is called GAUSS.

Note: Only one–dimensional integrals have been considered in this study. Further, interval analysis techniques, principal value integrals, parallel methods of numerical integration, monte carlo methods, number theoretic methods have also been excluded. The integrand is also assumed to be a function for which it is possible to write a FORTRAN/C subroutine/function i.e., integrands available as points on a grid or some other region are not considered.

2 The Methodology

The methodology adopted in this study is described below. Each of these is elaborated upon in the forthcoming sections.

1. Conduct performance evaluation of various algorithms for a variety of test problems.

2. Prepare a little pre–processing stage that extracts 'interesting' characteristics/features of the problem.

3. Develop an 'engine' that uses the results of the above two experiments to map from a given problem to an appropriate algorithm. In addition to performing this mapping, it is desirable if general heuristics can be drawn that might lead to some insight into the domain.

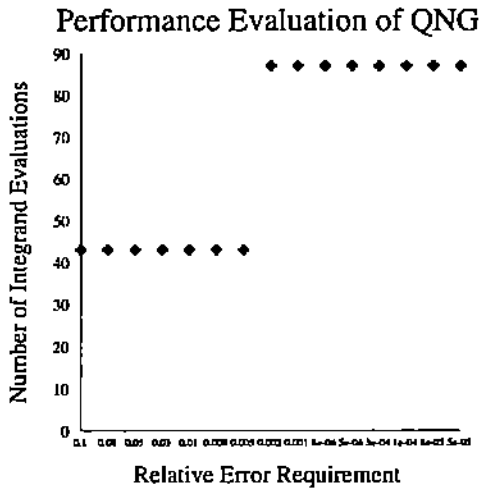

Figure 1: Performance of QNG for the sample integral evaluation

3 Performance Evaluation of Algorithms

The performance evaluation of algorithms is facilitated by packages like NAG, IMSL and QUADPACK. The GAMS category H2a indicates software meant for the evaluation of one dimensional integrals. 104 routines have been downloaded from various sites and these have been utilized in this study. A variety of test problems have been included in these implementations and several integrals with 'interesting' properties are also described in [1],[3],[7],[9]. Most of the test problems in [9] are parameterized; this gives rise to a huge number of test problems, numbering nearly 400.

(Some of the routines would not apply to all integrals, e.g., a certain gaussian routine would require that the integrand be expressed in the form $w(x)f(x)$ with $w(x)$ being some special weight function, such as a sinusoidal function.) For each routine and each applicable integrand, experiments were conducted with varying requirements on the relative error acccuracy ϵ_r. The strictest error requirement tried out was 10^{-8}. The data gathering module of GAUSS has been automated with shell scripts.

3.1 Sample Experiment

A particularly interesting example that is simple and illustrates the methodology is the integral

$$\int_0^1 x^{1/2} log(x) dx = \frac{4}{9}$$

. When the relative error requirement was reduced from 0.1 downto $3E-05$, the number of function evaluations varied as shown in Fig. 1.

It is to be noted that QNG is a simple non–adaptive automatic quadrature routin e that is based on a sequence of rules with increasing degrees of algebraic precision. As shown in fig. 1, an error requirement of 0.1 results in the evaluation of the integrand at 43 points. Incidentally, this

also satisfies all error requirements till 0.005. For a more stringent requirement of 0.002, we get 87 integrand evaluations which actually produces a relative error of $0.22E-04$. This explains the pattern of the graph above. Also, when we impose an error requirement of $1E-05$, we get the following O/P from the Performance Evaluation System (PES):

```
integral approximation =,-0.44444460E+00
Estimate of absolute error =, 0.22E-04
Number of function evaluations =    87
error code = 1
```

This error code (which was 0 for the previous error requirements), indicates that the maximum number of function evaluations has been achieved. For QNG, this means that the maximum number of 'steps' have been executed and that the integral is probably too difficult to be computed by this routine. For an adaptive routine, this means that the limit on the number of interval subdivisions has been achieved, which has been apriori set within the integrator.

We input all this information as predicate logic statements into the system. There is also a 'threshold predicate' which indicates the 'breakdown' point for a routine with a certain problem. In this manner, we take each integrand and impose varying requirements on the relative accuracy to determine the number of 'points'. The abrupt change in the above graph at a certain point is due to the non–adaptive nature of the integration routine. For adaptive routines, smoother transitions have been observed. (**Note:** It should be mentioned that graphs like the one above have not been translated to a 'slope–intercept' form as had been done earlier in PYTHIA. Instead, data points from the 'curves' have been directly input as predicate information.)

For example, the above information was coded as:

```
nfe(P1,QNG,0.1,43).
nfe(P1,QNG,0.05,43).
....
nfe(P1,QNG,0.002,87).
breakdown(P1,QNG,1E-05).
errorcode(P1,QNG,1E-05,1).
...
```

(**Note:** The nfe predicate models the number of function evaluations.) This information does not directly determine which routine is better for problem P1. To determine this, we introduce high–level rules which give rise to consequent predicates such as:

```
bestmethod(P1,QNG,0.1).
....
```

Various other error codes are possible from the routines – they could mean occurence of roundoff error, difficulties encountered in integrand behavior, divergent (or slowly convergent) integrals or a limiting number of cycles obtained. Many of these error codes are input to GAUSS as valuable information that it might (possibly) use to determine whether a particular routine is appropriate for a certain problem.

Special care also had to be taken if the integrand was not defined at one or more points in the integration interval. If the routine is not one that is particularly appropriate for singular integrals, then the function value had to be set equal to the limit of the function. Whenever this limit did not exist or was infinite, zero was substituted. Extensive literature on test problems ([1],[3],[7],[9]) helped to identify such problematic integrands.

Some more examples of test integrands are as follows:

Lyness's integral

$$I(\lambda) = \int_1^2 \frac{0.1}{(1-\lambda)^2 + 0.01} dx$$

Piessens' integrals

$$\int_0^1 x^\alpha log(\frac{1}{x}) dx = \frac{1}{1+\alpha^2}$$

$$\int_0^\pi cos(2^\alpha sin(x)) dx = \pi J_0(2^\alpha)$$

$$\int_0^\infty \frac{x^{\alpha-1}}{(1+10x)^2} dx = \frac{(1-\alpha)\pi}{10^\alpha sin(\pi\alpha)}$$

Note: The total number of test integrands utilized in this study was 286. This takes into account various parametrized functions such as the one above. Parameterized functions proved to be helpful because they encompass a family of functions with similar features and characteristics – this aids in the generalization of the system. The number of experiments thus rises to a staggering number (286 functions times 104 routines times 10 levels of accuracy = 446160). However, as mentioned before , many routines are not applicable to quite a few integands and results for a whole family of integrands can be quickly & automatically determined by shell scripts.

4 Automatic Feature Determination

A certain degree of automatic feature identification appears to be needed for this domain. Some of the more useful features are

- Whether the integrand can be expressed as $w(x)f(x)$ with several desirable features such as – $w(x)$ being one of several weight functions and f is smooth on $[a, b]$.
- Whether f is smooth
- Are there discontinuities of f in $[a, b]$?
- Are there singularities of the derivatives of f?
- Whether we know the location of the singularities of f.
- Whether we know the location of singularities of f'.
- Does f have end–point singularities?

- Does f have no singularities?
- Whether f exhibits an oscillatory behavior of non–specific type.
- Whether the range of integration is finite or infinite.

The first few characteristics can be symbolicaly determined from the rule–based interface to *Mathematica*. The latter features require the use of some kind of imagistic techniques or diagrammatic reasoning. We have utilized a primitive kind of method, drawing from earlier research on handwritten character recognition. The approach is as follows:

1. Draw the integrand in the domain of interest and plot it in a rectangular grid of 1000 by 1000 pixels.
2. 'Flood' a bit array of 1000x1000 based on the bit positions of the appropriate elements in the graphic. If a bit is lighted, enter a 1 in the corresponding position; else, enter a 0.
3. With this binary array of 1000x1000, compute the 'moment invariants' $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6$ [12]. These moment invariants are dimensionless quantities and serve to represent the image and they are invariant under translation, scaling and rotation operations. In this study, we are mostly interested in the first two transformations. Thus, only those moment invariants that are 'fixed' under translation and scaling are considered.
4. Having thus 'reduced' a function $f(x)$ to a few variables, we train a neural network to classify any new function into one of several families of functions. This is easily achieved with our current family of integrand functions.

The net effect of this is that we have some features that can be used to determine if a new function behaves in a manner similar to that of any previously seen function. Oscillatory behavior, Singularities and behavior of functions over infinite domains can be easily detected by this technique.

Thus, we have two mechanisms to achieve automatic feature identification:

- **The Imagistic component:** This determines the moment invariant features of function graphs and uses them to classify functions into several categories – a neural network maps the moment invariants to types of functions. For example, oscillatory functions were trained as a certain type, functions that have singularities in the domain of interest were trained as another type. The total number of categories tried out was 16 (8 families). These are:
  1. Logarithmic Functions
  2. Trigonometric Functions
  3. Exponential Functions
  4. Miscellaneous Transcendental Functions
  5. Functions with $x^{n/m}$ and $(a+bx)^m$
  6. Functions with (x^2-a^2) and x^m
  7. Functions with (a^2-x^2) and x^m

8. Other Algebraic Functions

Note: Each of the above classes includes functions that may or may not have singularities. Thus, each of the above is actually represented as two classes in the neural network – one which represents functions that have singularities and one without any singularities. This brings the actual number of classes to 16.

It was found that a high classification accuracy of 86.87% was achieved with the given data. The total number of functions that were 'plotted' were 800. To test the feasibility of the current approach, this set of 800 was initially split into two parts - 550 functions and 250 functions. The first set was used as the training set and the second part was used for testing. This gave rise to a generalization accuracy of 80% and a recall accuracy of 85.09%, bringing the overall accuracy to 83.44%. Having thus realized that the method yields sufficient accuracy, the system was 're-trained' with the whole set of 800 functions, which gave rise to the accuracy of 86.87%. (Most of the errors were caused by the neural network getting confused between logarithmic and exponential functions).

Example: The moment invariant features of certain functions that are quite similar are shown in the table below:

Function	$log \mid \phi_1 \mid$	$log \mid \phi_2 \mid$	$log \mid \phi_3 \mid$	$log \mid \phi_4 \mid$	$log \mid \phi_5 \mid$	$log \mid \phi_6 \mid$
1a	-6.746	-14.593	-26.353	-24.804	-50.846	-32.106
1b	-6.456	-14.219	-26.854	-24.855	-48.954	-32.667
2a	-5.881	-14.928	-20.305	-20.954	-41.585	-28.439
2b	-5.754	-14.636	-20.744	-20.665	-41.876	-28.897
3a	-6.178	-17.981	-23.524	-23.250	-46.692	-32.242
3b	-6.688	-17.010	-23.789	-23.011	-49.439	-32.878
4a	-6.134	-17.731	-22.136	-26.135	-53.565	-35.135
4b	-6.111	-17.197	-22.135	-26.315	-53.133	-34.789

It can be easily seen from the above table that figures 1a and 1b, 2a and 2b, 3a and 3b and 4a and 4b are classified as 'similar–looking' figures. As an example of the fruitfulness of this technique, the graphs of 1a and 1b are shown in Fig. 2. 1a represents the curve $y = x\sqrt{0.5 + 0.5x}$ and 1b is the curve $y = x^2\sqrt{0.5 + x}$.

It can be argued that this kind of imagistic reasoning does not reveal any information in addition to what can be determined symbolically. However, this idea has demonstrated that the same generic method can be used to 'evolve' clusters or families of functions. i.e., instead of specifying to the system what kind of function an image represents, we can make it figure this out for itself and create a new cluster if it deems that the function currently presented is one not encountered before. To this end, we can use the neuro–fuzzy clustering scheme recently developed [6].

- **The symbolic feature extraction component:** This is a module written in Mathematica. This is not yet complete but several important features can now be automatically ascertained.

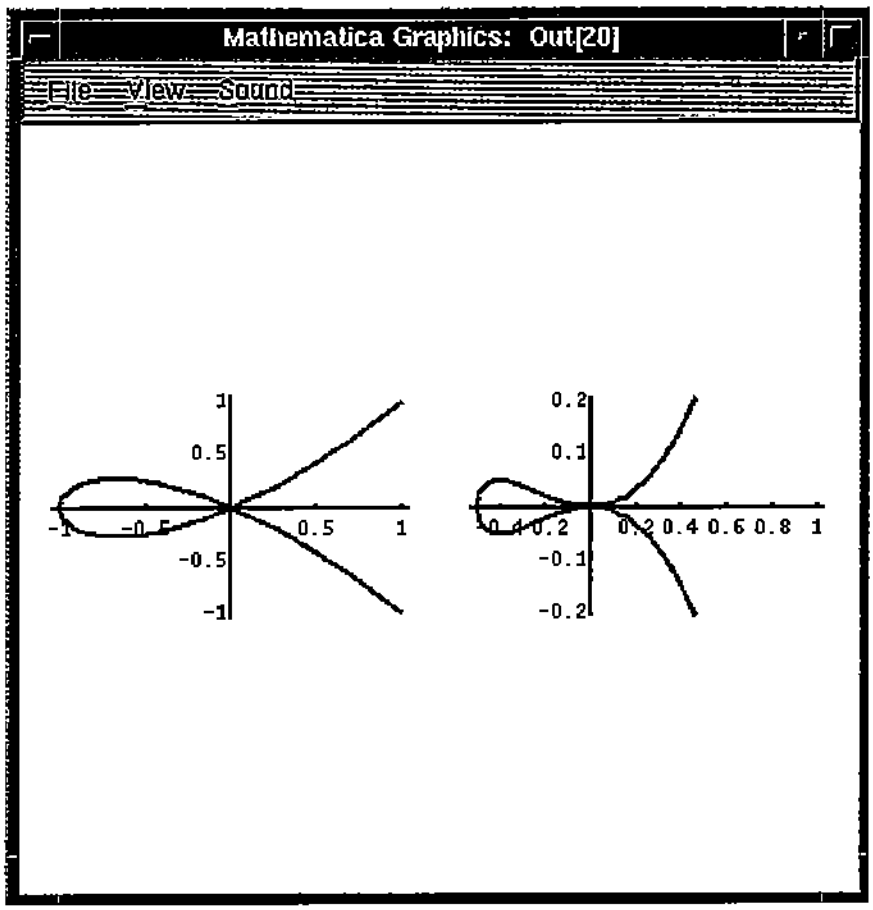

Figure 2: Plots of functions 1a and 1b

Advantage is taken of rule–based interface to Mathematica. This kind of methodology allows us to say things like:

```
Singularity[Times[f,g]] = Singularity[f] U Singularity[g]
Singularity[Plus[f,g]] = Singularity[f] U Singularity[g]
Singularity[Minus[f,g]] = Singularity[f] U Singularity[g]
Singularity[Times[x, Power[y, -1]]] = Singularity[f] U Zeros[g]
....
Singularity[Constant] = {}
```

In this way, the singularities of various functions can be expressed as set functions of singularities of their constituent functions all the way down to the 'basis' functions. Zeros of polynomials required by the above rules are determined from the root finder present in Mathematica. The root finder attempts to find the roots of the given expression using Newton's method and its variants.

Note: The above is not syntactically correct in Mathematica. The rules are described in this manner for the sake of clarity.

However, there are various occasions where the above rules fail to work, mainly because the root finder in Mathematica exits with an error code. Thus, a foolproof symbolic feature extraction technique is not yet ready. Whenever a rule fails, the features are determined by other means and input to the system. Currently the rules determine the following symbolic features of functions (with limitations):

- Whether the integrand can be expressed as $w(x)f(x)$ with several desirable features such as – $w(x)$ being one of several weight functions and f is smooth on $[a, b]$.
- Are there singularities of the derivatives of f?
- Whether we know the location of the singularities of f.
- Whether we know the location of singularities of f'.
- Does f have end–point singularities?
- Whether the range of integration is finite or infinite.

A slightly different approach to automatic feature identification is described in the section on future work.

As mentioned earlier, all features determined by these means are encoded as logic rules in GAUSS.

5 The "Expert" Methodology

The approach that is adopted in GAUSS is a highly knowledge intensive one. The paradigm of learning used here is inductive logic programming (ILP) [2]. This is a classical machine learning

technique that has recently seen a resurgence of interest, mainly due to reports of successful developments & applications. Background knowledge is represented as a set of predicate definitions and positive and negative examples. Given this, an ILP system attempts to construct a predicate logic formula so that all the positive examples can be logically derived from the background knowledge and no negative example can be logically derived. The ILP system used by GAUSS is Golem[8]. The advantages of such a system lie in the generality of representation of background knowledge.

6 Preliminary Results

The entire gamut of performance data gathered, and symbolic features extracted (or input by the 'user') have been coded as predicate logic rules. Sample rules extracted by Golem are:

```
method(Problem, DQAGK6) :-
        smooth(Problem),
        accuracy(Problem, -8),
        image(Problem, TYPE3),
        ~nosing(Problem),
        ~noderivsing(Problem),
        range(Problem, FINITE).

method(Problem, DQAGK1) :-
        smooth(Problem),
        accuracy(Problem, -8),
        image(Problem, TYPE2),
        sing(Problem),
        derivsing(Problem),
        range(Problem, FINITE).

method(Problem, QAGS) :-
        accuracy(Problem, Acc),
        image(Problem, TYPE2),
        sing(Problem),
        endptsing(Problem),
        noderivsing(Problem),
        range(Problem, FINITE).

method(Problem, QAGS) :-
        accuracy(Problem, Acc),
        image(Problem, TYPE6),
        sing(Problem),
        endptsing(Problem),
```

```
noderivsing(Problem),
range(Problem, FINITE).
```

This means that method DQAGK6 is most suitable to problem 'Problem' if Problem has behavior of TYPE3, the function is smooth, the range of integration is finite, there are no singularities of the function or its derivative and the accuracy level required is 10^{-8}. TYPE3 represents oscillatory behavior of a certain type and DQAGK6 is a simple globally adaptive QUADPACK routine (with the parameter KEY set to 6).

The last two rules are of particular interest – GAUSS has determined that the routine QAGS is good for any integrand with any error criterion if the function has end point singularities. True enough, TYPE2 & TYPE6 are types in imagistics that correspond to functions that have singularities and have logarithms and exponentials in them. Such rules have been found to be 'faithful' to the data gathered in this study.

Based on these rules, algorithm selection was conducted with the entire set of functions considered. An accuracy of 83% was observed.

Note: It is worth mentioning that the QUADPACK manual includes a ready reference table that allows a user to select the appropriate QUADPACK module for certain test problems. GAUSS has retrieved a rule quite similar a line of this table.

7 Notes

- The initial experiments seem to indicate favorably toward the use of ILP modules like Golem in systems like PYTHIA and GAUSS. The difference between our earlier approaches and this is that the newer method is highly knowledge intensive and, more importantly, uses relational descriptions of objects to influence the decision making process. In our earlier approaches to PYTHIA, the background knowledge about the domain could be expressed in only a limited form.

- As a related issue, the idea of conducting 'Scientific Discovery' in such mathematical domains can also be explored. This is similar to the work of Dr. Valdez Perez from CMU. For eg., the rules inferred by Golem can be examined to see if there are any simple (& obvious) rules that have been overlooked in the past.

- Packages like QUADPACK also include a decision–tree to help users choose the supplied algorithms. By restricting our attention to routines from only such packages, we can see if the rules inferred by our system have any relation to the decision tree models already formulated by the domain experts.

- The imagistic component can be enhanced and true 'discovery' of function classes (clustering) can be attempted. In the current schema, some more function classes may be introduced into the imagistic component such as:

  1. Trigonometric Functions with $asin^n(cx) + bcos^m(cx)$

2. Trigonometric Functions with $1 \pm sin^n(ax)$ and $1 \pm cos^m(bx)$
3. Trigonometric Functions with more complicated arguments
4. Functions with $(a^2 - x^2)^{1/2}$ and x^m
5. Functions with $(x^2 - a^2)^{1/2}$ and x^m
6. Functions with $(x^2 + a^2)^{1/2}$ and x^m
7. Inverse Trigonometric Functions
8. Hyperbolic and Inverse Hyperbolic Functions
9. Trigonometric Functions combined with powers of x
10. Exponential Functions combined with powers of x
11. Hyperbolic Functions combined with powers of x
12. Logarithmic Functions combined with powers of x

- The current algorithms are assumed to be 'monolithic' ones that are applied to the entire domain of integration. However, there exist occasions when a domain needs to be split up and different methods used to evaluate the integrals at different sub–intervals. Such kinds of rules can be included in the study.
- In research on similar lines, Sacks et.al., [11], have shown how their systems for intelligent scientific computing automatically construct numerical procdures through liberal use of high–order procedural abstractions. Such a kind of methodology would be appropriate for this domain. A potential system could 'chain' together necessary routines to determine features of functions. An agent based implementation is a possible means of achieving this task.
- Further work on symbolic analysis would lead to overcoming the pitfalls encountered currently.

References

[1] Bernsten, J. and Espelid, T. O. and Sorevik, T., *On the Subdivision Strategy in Adaptive Quadrature Algorithms*, J. Comput. Appl. Math. **35** (1991), 119–132.

[2] Bratko, I. and Muggleton, S., *Applications of Inductive Logic Programming*, Communications of the ACM **38** (November, 1995), no. 11, 65–70.

[3] Corliss, G. F. and Rall, L. B., *Adaptive, Self–Validating Numerical Quadrature*, SIAM J. Sci. Stat. Comput. **8** (1987), no. 5, 831–847.

[4] Davis, P. J. and Rabinowitz, P., *Methods of Numerical Integration*, Academic Press, Orlando, Florida, 1984, Second Edition.

[5] de Boor, C., *CADRE: An Algorithm for Numerical Quadrature*, Mathematical Software (Rice, J. R., ed.), Academic Press, New York, 1971, pp. 417–449.

[6] Joshi, A. and Ramakrishnan, N. and Rice, J.R. and Houstis, E.N., *A Neuro–Fuzzy Approach to Agglomerative Clustering*, Proc. ICNN'96, 1996.

[7] Lyness, J. N., *When Not to Use an Automatic Quadrature Routine*, SIAM Review **25** (January. 1983), no. 1, 63–87.

[8] Muggleton, S. and Feng, C., *Efficient Induction of Logic Programs*, Proc. of the First International Conference on Algorithmic Learning Theory (Arikawa, S. and Goto, S. and Ohsuga, S. and Yokomori, T., ed.), Japanese Society for Artificial Intelligence, Tokyo, 1990, pp. 368–381.

[9] Piessens, R. and de Doncker–Kapenga, E. and Überhuber, C. W. and Kahaner, D. K., *Quadpack*, Springer–Verlag, New York, 1983.

[10] Rice, J. R., *Methodology for the Algorithm Selection Problem*, Performance Evaluation of Numerical Software (L. D. Fosdick, ed.), North–Holland, 1979, pp. 301–307.

[11] Sacks, E.P. et.al., *Intelligence in Scientific Computing*, Communications of the ACM **32** (May, 1989), no. 5, 546–562.

[12] Wood, J., *Invariant Pattern Recognition: A Review*, Pattern Recognition **29** (January, 1996), no. 1, 1–17.