ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 2, Issue 2, Ver. 2 (April - June 2014)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

# Comparative Study of Complexity of Algorithms For Ordinary Differential Equations

[I]D.S. Ruhela, [II]R.N.Jat
[I]Dept. of Computer Application, S.M.L. (P.G.) College, Jhunjhunu (Raj.), India
[II]Dept. of Mathematics, University of Rajasthan, Jaipur, India

**Abstract**
*Analysis of an algorithm is to determine the amount of resources such as time and storage necessary to execute it. The efficiency or complexity of an algorithm is based on the function relating the input length to the number of operations to execute the algorithm. In this paper, the computational complexities of different algorithms used in ordinary differential equations numerical methods are analyzed.*

**Keywords**
*Time Complexity, Algorithms for Numerical Methods, Order, Big Oh, Adaptive Stepwise Method*

## I. Introduction

Since the time of Leibniz (1646-1719) and Newton (1642-1727), many mathematical models in engineering, physics and finance have been formulated using deterministic or stochastic differential equations. The rapid development of high speed computers over the last decades accelerates the possibilities of efficiently utilizing these models. To use mathematical models on computers we need numerical approximations of solutions to differential equations. As for all reliable measurements, a useful computation requires an estimate of its accuracy. In general, higher accuracy can only be achieved by increasing the computational work.

An algorithm is a list of instructions that solves every instance of a problem in a finite number of steps. Analyzing an algorithm determines the amount of "time" that the algorithm takes to execute on a particular input size. Here, the amount of time means an approximation of the number of operations that an algorithm performs. For example, if the input to an algorithm is a graph, the number of nodes and edges in the graph can describe the input size. The key idea to measure time and space as a function of the length of the input came in the early 1960's by Hartmanis and Stearns and thus computational complexity research started. In the analysis of algorithms, it is not important to know exactly how many operations an algorithm does. The general behavior i.e. the overall growth rate of the algorithm is an important factor. The growth rate of the algorithm is the rate of increase in operations for an algorithm as the size of the problem increases. As the rate of growth of an algorithm is determined by the largest term in an equation, the terms that grow more slowly are discarded Big oh to determine the computational complexity. The computational complexities of all the algorithms used for ordinary differential equations using iterative methods discussed in this paper are determined, using Big Oh notation.

## II. Asymptotic Notation

We have clearly gone into detail for evaluating the running time of such simple algorithm. Such an approach would clearly prove cumbersome if we had to perform it for more complicated algorithms. In general, each step is pseudo-code description and each statement in a high level language implementation corresponds to small number of primitive operations that does not depend on input size. Thus we can perform a simplified analysis that estimates the number of primitive operation executed up to a constant factor, by counting the steps of pseudo-code or the statements of high level language executed. Fortunately, there is a notation that allow us to characterize the main factor affecting an algorithm's running time without going into all detail of exactly how many primitive operations are performed for each constant time of instructions

Adaptive methods are useful to combine the two goals of controlling the approximation error and minimizing the computational work. There are numerous adaptive algorithms for differential equations, however the theoretical understanding of optimal convergence rates of adaptive algorithms is not as well developed. This work studies adaptive algorithms and their convergence rates for ordinary differential equations.

The main ingredient of the adaptive algorithms here is an error expansion of the form:

Global error = local error • weight + higher order error, (3.1)

With computable leading order terms. Here by computable we mean that the leading order error depends only on approximate solution, in other words, the leading order terms are in a posteriori form, in contrast to an a priori error estimate based on the unknown exact solution. For a deterministic case, such error expansion for differential equations can be derived by three different methods. Firstly, the classical error equation based on linearization derives the error expansion by (Henrici, 1962; Harrier et al., 1993; Dahlquist and Bork, 1974; Dahlquist and Bork, 2003). The Galerkin orthogonally using either local problems or the residual also leads to (3.1), see the survey papers (Eriksson et al., 1995; Oden, 1997; Becker and Rannacher, 2001). Finally, the error expansion (3.1) can be derived by the variation principle following (Alekseev, 1961) and (Gobner, 1967). For weak approximations of SDEs, the work (Talay and Tubaro, 1990) develops an error expansion in a priori form under assumptions of sufficient smoothness by the Euler method. Then (Bally and Talay, 1996) extends this error expansion to non-smooth functions using the Malliavin calculus. (Gobet, 2000; Gobet, 2001) extends the results of Talay and Tubaro to killed and reflected diffusion. Based on stochastic flows and discrete dual backward problems, (Szepessy et al., 2001) derives new expansions of the computational error with computable leading order term in a posteriori form. General construction and analysis of the convergence order for approximations of SDEs can be found in (Kloeden and Platen, 1992).Based on these error estimates, the goal of adaptive algorithms for differential equations is to solve problems with an adapted mesh using as little computational work as possible, for a given level of accuracy. Then a typical adaptive

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

algorithm does two things iteratively:

(1) If the error indicators satisfy an accuracy condition, then it stops; otherwise

(2) The algorithm chooses where to refine the mesh and re computes the error indicators then makes an iterative step to (1).

Therefore the indicators not only estimate the localization of the global error but also give information on how to refine the mesh in order to achieve optimal efficiency.

An approximation of the weight function in (1.1), plays an important role for step (2). The weight function solves a certain linear backward dual problem obtained by linearizing a given forward problem around the approximate solution.

Therefore, the algorithm uses additional computational work to obtain the error estimate, which also informs where to refine the mesh to achieve optimal efficiency. The use of dual functions is well known for adaptive mesh control for ODEs and PDEs, see (Babuska and Miller, 1984a; Babuska and Miller, 1984b; Babuska and Miller, 1984c; Johnson, 1988; Johnson and Szepessy, 1995; Estep, 1995; Becker and Rannacher, 1996). Despite the wide use of adaptive algorithms and the well-developed theory of a posteriori error estimates, less is known theoretically on the behavior of adaptive mesh refinement algorithms. However, there are recent important contributions;

The work (DeVore 1998) studies the efficiency of nonlinear approximation of functions, including wavelet expansions, based on smoothness conditions in Besov space. Inspired by this approximation result, (Cohen et al., 2000) proves that a wavelet based adaptive N-term approximation produces an approximation with asymptotically optimal error in the energy norm for linear coercive elliptic problems. Then the work (Binev et al., 2002) and (Stevenson, 2003) extend the ideas of (Morin et al., 2000) to prove similar optimal error estimates in the energy norm for piecewise linear finite elements applied to the Poisson equation, see also (Dorfler, 1996). The modification includes a coarsening step in the adaptive algorithm to obtain bounds on the computational work. For solving ODEs, (Stuart, 1997) and (Lamba and Stuart, 1998) prove the convergence of ODE23 of MATLAB version 4.2. For strong approximation of SDEs, the work (Hofmann et al., 2000; Hofmann et al., 2001) and (Muller-Gronbach, 2000) prove optimal convergence rates of adaptive and uniform step size control. These topics can be linked to information based complexity theory, see (Bakhvalov, 1971; Werschulz, 1991; Novak, 1996; Traub and Werschulz, 1998).

## III. The "Big-Oh" Notation

Let f (n) and g (n) be function mapping nonnegative integers to real numbers. We say that f (n) is O (g (n)) if there is a real constant c>0 and an integer constant n0>=1 such that f (n) <=cg (n) for every integer n>=n0. This definition is referred to as "big-Oh" notation. Alternatively, we can also say "f (n) is order g (n) [2].

## IV. Euler's Method

The Euler's method is not very accurate. To obtain a numerical solution with an acceptable accuracy, we have to use a very small step size h. A small step size h implies a larger number of steps, thus more computing time. An ordinary differential equation of order one can be represented as: dy/ dx = f (x) –– (1) with initial conditions x=x0 and y=y0 Let y=f(x) be the solution of (1).

The curve y=f(x) can be shown in a graph. Consider a point "P" on the curve y=F(x) whose coordinate(x0, y0), draw a tangent on the curve at point P. Further consider a point on the axis of x at a distance h=(x1-x0), from x0. Suppose the coordinate of this point meets the tangent by y1. Equation of tangent at P (x0, y0).Y-y0= (dy/dx) x0, y0(x=x0)

By (1) slope of the tangent at p is (dy/dx) x0, y0=f(x0, y0)

Y-y0=f(x0, y0) (x-x0)

The point R(x1, y1) lies on the tangent (2)

Y1-y0=f(x0, y0) (x1-x0)

Y1=y0+hf(x0, y0)

Y1 gives the coordinate of tangent at x=x1 this coordinate is nearly equal to coordinate f curve provider h is small. Similarly y2=y1+h*f(x1, y1)

Yn=yn-1+h*f(xn-1,yn-1) which is Euler's formula function f(x,y) is approximated by f(x0,y0).

### A. Algorithm for Euler's Method

To find the solution of ODE dy/dx = f (x) when the solution at the initial point x=x1 is given, and the solution is desired in the interval [x1, x f]

1. Begin
2. Read x1, y1, xf, h
3. Set x=x1
4. Set y=y1
5. Set i=1
6. Write i,x,y
7. While(x<=xf) do

I. Set y=y+h* f(x, y)

II. Set x=x+h

III. Set i=i+1

IV. Write i, x, y

End while

8. End

### B. Counting Primitive Operation

1. Reading the value of variable Read x1, y1, x, h contributes four unit of count.
2. Assigning values x=x1, y=y1 and i=1 contributes three unit of count.
3. Writing i, x, y contributes three unit of count.
4. Before entering the body of loop condition (x<=xf) is verified. This action corresponds one primitive operation.

At the beginning of Set y=y+h*f(x,y), x=x+h and i=i+1 are calculated. This action correspond to executing eight primitive operation

To write i, x, y contributes three operations. These operations performed n times.

To summarize, the number of primitive operations t (n) executed by algorithm is at least.

T (n) = 4+3+3+12n

T (n) =12n+10.

So the complexity of algorithm used for Euler's method is O (n).

In fact, any polynomial $a^k n^k + a^{k-1} n^{k-1} \ldots\ldots + a^0$ will always be O ($n^k$)

Using example dy/dx=1-y, where y0=0 at x0=0 in the range 0<=x<=4 using h=.1

(Using program code in C Language….)

Table - 1: No of operations to find value of y for given value of x

| X | .1 | 1.0 | 2.0 | 3.0 | 4.0 |
|---|---|---|---|---|---|
| Y | .095238 | .632426 | .864889 | .950337 | .981747 |
| No of Iterations | 4 | 4 | 3 | 3 | 2 |

## V. Runge-Kutta Second Order Method

Second-order Runge-Kutta methods are obtained by using a weighted average of two increments y(x0), k1 and K2. For this equation

Yn+1=yn+ak1+bk2

K1=h*f (xn, yn)

K2=h*f (xn+αh, yn+β k1)

### A. Algorithm for Runge-Kutta second order Method

To find the solution of ODE, when the solution at the initial point x=x1 is given, and the solution is desired in the interval [xl, xf]

1. Begin
2. Read x1, y1, xf, h
3. Set x=x1
4. Set y=y1
5. Set i=1
6. Write i, x, y
7. While(x<=xf) do
Set k1=f(x, y)
Set x=x+h
Set K2=f(x, y+k1)
Set y=y+ (k1+K2)/2
Set i=i+1
Write i, x, y
End while
8. End

### B. Counting Primitive Operation

1. Reading the value of variable Read x1, y1, xf, h contributes four unit of count.
2. Assigning values. x=x1, y=y1 and i=1 contributes three unit of count.
3. Writing i, x, y contributes three unit of count.
4. Before entering the body of loop condition (x<=xf) is verified. This action corresponds one primitive operation.

At the beginning of x=x+h, K2=f(x, y+k1), y=y+ (k1+K2)/2, i=i+1 are calculates. This action correspond to executing nine primitive operation

To write i, x, y contributes three operations. These operations performed n times.

To summarize, the number of primitive operations t (n) executed by algorithm is at least.

t (n) = 4+3+4+13n

t(N)=13n+10.

So the complexity of algorithm used for Runge-kutta second order method is O (n).

In fact, any polynomial $a^k n^k+a^{k-1}n^{k-1}\ldots\ldots+a^0$ will always be O ($n^k$)

Using example dy/dx=1-y, where y0=0 at x0=0 in the range 0<=x<=4 using h=.1 (Using program code in C Language….)

Table - 2 : No of operations to find value of y for given value of x

| X | .1 | 1.0 | 2.0 | 3.0 | 4.0 |
|---|---|---|---|---|---|
| Y | .095000 | .631659 | .864178 | .954699 | .983305 |
| No of Iterations | 4 | 4 | 3 | 3 | 2 |

## VI. Runge-Kutta Fourth Order Method

That is the basic idea of the Runge-Kutta method (Abramowitz and Stegun, and Gear ), give various specific formulas that derive from this basic idea. By far the most often used is the classical fourth-order Runge-Kutta formula, which has a certain sleekness of organization about it:

k1 = h*f (xn; yn)

k2 = h*f (xn +h/2, yn + k1/2)

k3 = h*f (xn +h/2; yn + K2 /2)

k4 = h*f (xn + h/2; yn + k3)

$y_{n+1} = y_n$ +k1/6+k2/3+k3/3+k4/6+ O($h^5$)

The fourth-order Runge-Kutta method requires four evaluations of the right hand side per step. This will be superior to themed point method if at least twice as large a step is possible with for the same accuracy. Is that so? The answer is: often, perhaps even usually, but surely not always! This takes us back to a central theme, namely that high order does not always mean high accuracy. The statement "fourth-order Runge-Kutta is generally superior to second-order" is a true one, but you should recognize it as a statement about the contemporary practice of science rather than as a statement about strict mathematics. That is, it reflects the nature of the problems that contemporary scientists like to solve.

### A. Algorithm for Runge-Kutta fourth order

1. Read x1, y1, xf, h
2. X=x1;
3. Y=y1;
4. i=1;
5. Write i, x, y
6. While(x1<=xf)
k1=h*f(x1, y1);
k2=h*f(x1+h/2, y1+k1/2);
k3=h*f(x1+h/2, y1+k2/2);
k4=h*f(x1+h, y1+k3);
k= (k1+2*K2+2*k3+k4)/6;
Write, k1, K2, k3, k4, k
y1=y1+k;
X1=x1+h;
i=i+1;
Write i, x1, y1
End while
7. End

### B. Counting Primitive Operation

1. Reading the value of variable Read x1, y1, xf, h contributes four unit of count.
2. Assigning values x=x1, y=y1 and i=1 contributes three unit of count.

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

3. Writing i, x, y contributes three unit of count.
4. Before entering the body of loop condition (x<=xf) is verified. This action corresponds one primitive operation.
At the beginning of k1=h*f(x1, y1) is calculated and count three operation
k2=h*f(x1+h/2, y1+k1/2) is calculated and count seven operation.
, k3=h*f(x1+h/2, y1+k2/2) is calculated and count seven operation.
k4=h*f(x1+h, y1+k3) is calculated and count five operation. k= (k1+2*K2+2*k3+k4)/6 is calculated and count six operation.
Writing K1, K2, K3, K4, K contributes five unit of count.
y1=y1+k, x1=x1+h, i=i+1 contributes six unit of count.
To write i, x, y contributes three operations. These operations performed n times.
To summarize, the number of primitive operations t (n) executed by algorithm is at least.
t (n) = 4+3+4+34n
=34n+10.
So the complexity of algorithm used for Runge-kutta fourth order method is O (n).
In fact, any polynomial $a^k n^k+a^{k-1}n^{k-1}……+a^0$ will always be O ($n^k$)
Using example dy/dx=1-y, where y0=0 at x0=0 in the range 0<=x<=4 using h=.1

Table - 3: No of operations to find value of y for given value of x

| X | .1 | 1.0 | 2.0 | 3.0 | 4.0 |
|---|---|---|---|---|---|
| Y | .095163 | .632120 | .864664 | .953121 | .981664 |
| No of Iterations | 4 | 4 | 3 | 3 | 2 |

## VII. Adaptive Step size Numerical Methods

In this we analyze the problem of adaptivity for one step numerical methods for solving ODE with a view to generating minimal cost for which the local error is below a prescribed tolerance. Using the adaptive iteration for h it is possible to make the use of the algorithm more computationally effective. The complexity of the algorithm for the method defined by [Goldberg 2007 can be calculated as:

### A. Algorithm using adaptive step size

To find the solution of ODE dy/dx=f(x), when the solution at the initial point x=x1 is given, and the solution is desired in the interval [xl, x f] with initial interval .1 and .05 respectively.
1. Begin
2. Read x1, y1, xf
3. Set x0=x1
4. Set y0=y1
5. Set h1=.1
6. Y1=y0+h1*f(x0, y0)
7. X1=x0+1
6. I=3
7. While(x<=xf) do
I. h[i]i=.9*(h [i-1]-h [i-2])/(y[i-1]-y[i-2])
Ii. Set y[i] =y[i] +h[i]* f(x [i-1], y [i-1])
IIi. Set x=x+1
IV. Write i, x, y 1V i=i+1
End while
8. End

### B. Counting Primitive operation

1. Reading the value of variable Read x1, y1, xf contributes three unit of count.
2. Assigning values x0=x1, y0=y1 and h=.1 contributes three unit of count.
3. Writing i, x, y contributes three unit of count.
4. Calculating x1, y1 and initialization i=3 contributes four units.
4. Before entering the body of loop condition (x<=xf) is verified. This action corresponds one primitive operation.
At the beginning of loop h[i], y[i] and x[i] calculated. This action correspond to executing ten primitive operation
To write i, x, y contributes three operations. These operations performed n times.
To summarize, the number of primitive operations t (n) executed by algorithm is at least.
t (n) = 3+3+3+4+10n+3
t (n)=10n+16 .
So the complexity of algorithm used for Adaptive step size method is O (n).
In fact, any polynomial $a^k n^k+a^{k-1}n^{k-1}……+a^0$ will always be O (nk)
Using example dy/dx=1-y, where y0=0 at x0=0 in the range 0<=x<=4 using h=.1: Let us choose the initial step size h1=.1 and h2=.05 using adaptive recurrence with coefficient q=.9. The first approximation of y(t) are the same
X=1 y=.632 using h1=.1 and x=2 y=.865 using h2=.05
H3=.9*(.1-.05) x.001/ (.865-, 632)
=.000193
At X=3 y=.950 using h3=.000193 using two iteration
Now h4=.9x (.05-.000193)x.001/(.950-.865)
=.0000527
At x=4 y=.982 using h4=.0000527 using single iteration.

Table 4 : No of operations to find value of y for given value of x

| X | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Y | .632 | .865 | .950 | .982 |
| No of iterations | 4 | 2 | 2 | 1 |

## VIII. Conclusion

The following table Shows the value of y at give value of x for f(x,y)=1-y with initial value y=0 at x=0 and execution time for code of program:

Table - 5: Comparison for number of iteration and execution time for various functions.

| | X=1 | X=2 | X=3 | X=4 |
|---|---|---|---|---|
| Euler's | Y=0.632<br>Time=0.071272<br>Iterations=6 | Y=0.865<br>Time=0.124469<br>Iterations=5 | Y=0.950<br>Time=0.158581<br>Iterations=3 | Y=0.982<br>Time=0.173595<br>Iterations=4 |
| RungaKutta 2nd Order | Y=.631459<br>Time=0.003992<br>Iterations=4 | Y=0.864178<br>Time=0.007928<br>Iterations=4 | Y=0.949944<br>Time=0.018575<br>Iterations=4 | Y=0.981552<br>Time=0.030541<br>Iterations=3 |
| RungaKutta 4th Order | Y=.670844<br>Time=.000378<br>Iterations=4 | Y=0.89165<br>Time=0.00537<br>Iterations=4 | Y=0.964338<br>Time=.005384<br>Iterations=4 | Y=0.988262<br>Time=0.009613<br>Iterations=3 |
| Adaptive Method | Y=,632<br>Time=.071272<br>Iterations=4 | Y=0.877914<br>Time=0.000843<br>Iterations=2 | Y=0.950295<br>Time=0.004714<br>Iterations=2 | Y=0.981614<br>Time=0.006600<br>Iterations=1 |

In order to compare the Adaptive Method with Euler's method, Runge- Kutta second order method and Rungekutta fourth order a variety of functions are used for with same accuracy. The time complexity of Euler's method is O(n), RungeKutta method is and O(n), RungeKutta method is O(n) . The table-5 shows that for function f(x)=1-y the number of iteration for x=4 is 4, RungeKutta method is 3 while in Adaptive method number of iteration is 1 for same value of x=4. The execution time of Adaptive method is smallest among all .Hence the execution time for Adaptive method is less and theprogram runs faster.

## References

[1] *Alekseev, V."An estimate for the perturbations of the solutions of ordinary differential equations" ii. Vestnik Moskov. Univ. Ser. I Mat. Mech, 3:3–10.1961*

[2] *Babuska, I. and Miller, A." The post-processing approach in the finite element method, i: calculations of displacements, stresses and other higher derivatives of the displacements." Int. J. Numer. Meth. Eng., 20:1085–1109. 1984a*

[3] *Babuska, I. and Miller," A. The post-processing approach in the finite element method, ii: the calculation of stress intensity factors."Int. J. Numer. Meth. Eng., 20:1111–1129.1984b.*

[4] *Babuska, I. and Miller, A, "The post-processing approach in the finite element method, iii: a posteriori error estimation and adaptive mesh selection" . Int. J. Numer. Meth. Eng., 20:2311–2324.1984c*

[5] *Bakhvalov, N, "On the optimality of linear methods for operator approximation in convex classes of functions."USSR Comput. Math. and Math. Phys., 11:244–249.. 1971*

[6] *Bally, V. and Talay, D, "The law of the Euler scheme for stochastic differential equations i: Convergence rate of the distribution function". Probably. Theory Related Fields, 104(1):43–60, 1996*

[7] *Becker, K. and Rannacher, R., "A feed-back approach to error control in finite element methods: basic analysis and examples". East-West J. Numer. Math., 4(4):237–264.1996.*

[8] *Becker, K. and Rannacher, R, "An optimal control approach to a posteriori error estimation in finite element methods". Acta Numerica, 10:1–102. 2001.*

[9] *Binev, P., Dahmen, W., and DeVore, R.," Adaptive finite element methods with convergence rates" . Technical report, Dept. of Math., Univ. of South Carolina.2002.*

[10] *Buchman, F., "Computing exit times with the Euler scheme. Technical report", ETH, Zurich, Switzerland. 2003.*

[11]. *DeVore, R, "Nonlinear approximation". Acta Numerica, pages 51–150. . 1998.*

[12]. *Eriksson, K., Estep, D., Hansbo, P., and Johnson, C. "Introduction to adaptive methods for differential equations", Acta Numerica, pages 105–158, 1995.*

[13] *Estep, D, "A posteriori error bounds and global error control for approximation of ordinary differential equations."SIAM J. Numer. Anal., 32:1–48, .1995.*

[14] *Gobet, E., "Weak approximation of killed diffusion using Euler schemes. Stochastic"Process. Appl., 87(2):167–197, 2000*

[15] *Gobet, E., "Euler schemes and half-space approximation for the simulation diffusion in a domain". ESAIM probably. Statist. 5:261–297. 2001*

[16] *Harrier, E., Nørsett, S., and Wanner, G. , "Solving Ordinary Differential Equations" I. Springer-Verlag.(1993).*

[17] *Henrici, P. , "Discrete Variable Methods in Ordinary Differential Equations." John Wiley & Sons Inc. 1962*

[18] *Hofmann, N., Muller-Gronbach, T., and Ritter, K., "Optimal approximation of stochastic differential equations by adaptive step-size control". Math. Comp., 69:1017–1034.2000*

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

*[19] Hofmann, N., Muller-Gronbach, T., and Ritter, K., "The optimal discretization of stochastic differential equations". J. Complexity, 17:117–153. 2001*

*[20] Johnson, C, "Error estimates and adaptive time-step control for a class of one-step methods for stiff ordinary differential equations". SIAM J. Numer Anal., 25:908–926. 1988*

*[21] Johnson, C. and Szepessy, A."Adaptive finite element methods for conservation laws based on a posteriori error estimates."Comm. Pure Appl. Math., 48:199–234.1995*

*[22] Kloeden, P. and Platen, E."Numerical solution of stochastic differential equations. Applications of mathematics" 23, Springer–Verlag. 1992*

*[23] Lamba, H. and Stuart, A. M., "Convergence results for the mat lab ode23 routine. "BIT, 38(4):751–780. 1998*

*[24] Morin, P., Nochetto, R. H., and Siebert, K. G."Data oscillation and convergence of adaptive fem. "SIAM J. Numer. Anal., 38(2):466–488.(2000).*

*[25] Novak, E., "On the power of adaption."J. Complexity, 12:199–237. 1996*

*[26] Stevenson, R. . . "An optimal adaptive finite element method. Technical report" ,Univ. of Utrecht. Reprint no. 1271. 2003*

*[27] Strauss, W., "Partial differential equations: An introduction." John Wiley & Sons, Inc.1992*

*[28] Stuart, A. "Probabilistic and deterministic convergence proofs for software for initial value problems."Numerical Algorithms, 14:227–260. 1997*

*[29] Szepessy, A., Tempone, R., and Zouraris, G. E., "Adaptive weak approximation of stochastic differential equations". Comm. Pure Appl. Math., 54(10):1169–1214.2001*

*[30] Talay, D. and Tubaro, L., "Expansion of the global error for numerical schemes solving stochastic differential equations."Stochastic Anal. Appl., 8:483– 509.1990*

*[31] Traub, J. and Werschulz, A., "Complexity and Information." Cambridge University Press, Cambridge.1998*

*[32] Werschulz, A., "The Computational Complexity of Differential and IntegralEquations, an Information-Based Approach." The Clarendon Press, Oxford University Press, New York.*