The binary floating point digital differential analyzer

by J. L. ELSHOFF and P. T. HULINA

The Pennsylvania State University
University Park, Pennsylvania

INTRODUCTION

Twenty years ago the digital differential analyzer, DDA, was developed to replace the analog computer in the solution of differential equations. Although the DDA is slower than the analog computer, the DDA is capable of more accurate results since its accuracy is not bounded by its component characteristics. The cost of solving differential equations with the DDA is quite low compared with other methods such as a general purpose machine, since the DDA is a more simple device.

As time has passed, advances have been made in DDA technology. These advances have resulted in increased speed and accuracy,[1,2] reductions in cost,[1] and improvements in man-machine interface.[3] Still the DDA is seldom used except as a special purpose device. Despite the dependence of the problem solution on the quality of the components and the higher cost of the analog computer, analog computation continues to grow in popularity. Similarly, general purpose computers continue to be more widely used even though they cost more and solve differential equations at a slower rate than the DDA.

In recent years analog and digital computers have been combined into hybrid systems.[3,4] In theory, the hybrid system takes advantage of the high speed of the analog computer and the easy programmability and decision capabilities of the digital computer. In practice, however, the speed of the analog computer is greatly reduced in operational performance by digital software and the digital-to-analog and the analog-to-digital conversion hardware. The general purpose digital computer can be programmed in an easy problem-oriented language like Fortran while the analog portion of the problem must be physically patched.

This paper concerns itself with a brief review of DDA technology and an investigation of ways in which to expand that technology. The emphasis is placed on increasing the speed, reducing the cost, and improving the utility of the DDA in such a way that the DDA would replace the analog computer and provide a more practical hybrid system.

THE DDA

The vector form of the general linear homogeneous constant coefficient ordinary differential equation can be written

$$\dot{x} = Ax, \qquad x(0) = x_0 \tag{1}$$

where A is a constant $m \times m$ matrix, and x and $\dot{x}$ are $m \times 1$ column vectors. In rectangular integration the vector difference equation that replaces equation (1) is

$$y(n+1) = y(n) + \dot{y}(n)\Delta t = y(n) + \Delta y(n) \tag{2}$$

where from equation (1)

$$\Delta y(n) = \dot{y}(n)\Delta t = (\Delta t)Ay(n).$$

For any given value of $y(0)$, an iterative solution of equation (2) can be obtained. If $y(0) = x_0$, then $x(t)$ is approximated through the relation

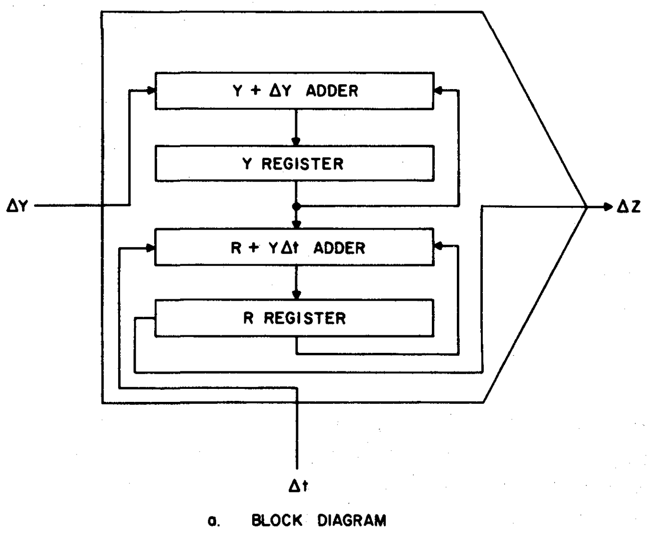
$$x(n\Delta t) = y(n) + 0((\Delta t)^2)$$

where $0((\Delta t)^2)$ represents the truncation error and $n\Delta t = t$.

In a DDA the fractional part of $y(n)\Delta t$ is held in a residue register (R register) and only the integer part is used in equation (2). Let $R(n)$ be the contents of the R register at $t = n\Delta t$. Then the equation for $y(n)\Delta t$ is modified to

$$y(n)\Delta t + R(n-1) = \Delta Z(n) + R(n)$$

where $\Delta Z(n)$ is a signed integer and $|R(n)| < 1$. Bartee, Lebow, and Reed,[5] Huskey and Korn,[6] and



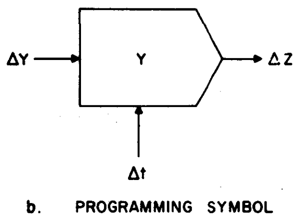


Figure 1—Basic DDA integrator

McGhee and Nilsen[2] explain the mathematical principle of the DDA in detail.

Figure 1 displays the block diagram and a programming symbol for the DDA integrator. The Y register contains the value of $y(n)$. The ΔY input holds the value of Δy, the incremental change of the integrand, at each step. The ΔZ output and R register contain the integral and residue of $y(n)\Delta t$, respectively. Finally, the Δt input holds the value of Δt, the step size of the independent variable.

With these values available, the iterative solution to equation (2) can be completed. A single step of rectangular integration is performed by the transfer equations

$$\text{Integration phase: } Y\Delta t+R\rightarrow\Delta Z+R$$

$$\text{Incrementation phase: } Y+\Delta Y\rightarrow Y$$

where ΔY is a weighted sum of ΔZ outputs. By alternating the integration and incrementation phases, the solution to equation (2) is iteratively realized.

In operation the inputs and outputs in a DDA are represented by two binary bits, the sign bit and coefficient bit. For an arbitrary problem the input and output increments each represent a fixed magnitude. For example, if $\Delta Y=c$, where c is a constant, then the coefficient bit is one or zero depending on whether or not there is a ΔY during each particular incrementation phase. The sign bit is one or zero depending on whether ΔY is negative or positive. Thus, the value c is fixed for the problem during the programming and is not actually transferred during the solution. Only a signed coefficient of one or zero is transferred.

In practice the inputs and outputs in a base e DDA are coefficients of incremental values that are equal to integer powers of e. By choosing a step size equal to 2^{-i} in a binary DDA the $Y\Delta t$ term can be calculated by a simple shift instead of a multiplication. Let $\Delta t=2^{-i}$ for i a positive integer, then the Y register is assumed to be shifted right i positions so that the R and Y registers have their bits aligned. Thus, the integration phase is reduced to a simple addition.

The method of programming a DDA resembles that of programming an analog computer. A certain quantity is assumed to be known. The other values are calculated from the assumed value. Finally, the assumed value is derived back from the known values. The actual program must then be physically patched.

The fixed point arithmetic used in the DDA is a major disadvantage of the DDA. Problems must be magnitude scaled for solution. Although Gill[7] and Knudsen[8] have developed a completely systematic procedure for scaling a DDA, the scaling problem is difficult and solution accuracy depends upon estimated maximum values.

Since the DDA has not enjoyed widespread use, most of the developments in DDA's have been pointed at particular problem solutions. Usually emphasis is placed on increasing the speed and accuracy of the DDA, which happen to be inversely proportional. An improvement in one aspect of DDA's is often compromised by added complications and new problems in other aspects. Like the analog, the inconvenience in using the DDA contributes to its lack of popularity. Because of the lack of popularity, only slight attention has been focused on improving user convenience.

The emphasis in this work was aimed at user convenience in a hybrid computing system. Since the major problems seemed to be in programming and scaling, they were given a high priority. Being digital, two components can be connected or disconnected by passing their connection time through an AND gate with a switching variable that is either on or off, respectively. Thus, in a hybrid system, the general purpose computer can be used to program the DDA. The obvious answer to the scaling problems seemed to be floating point arithmetic. The use of floating point arithmetic was also expected to be more accurate which is a very desirable effect. Floating point arithmetic was implemented in a DDA design and the design simulated on a general purpose digital computer. The implementation and simulation results appear in the remainder of this paper.

THE BINARY FLOATING POINT DDA

The purpose of this section is to present the binary floating point digital differential analyzer (BFPDDA). The BFPDDA differs from the conventional DDA in that the incremental units being transferred between the components are exponents. Multiple bits must be used to transmit an exponent instead of the usual one or two transmission bits in the regular DDA. Yet with as few as seven bits, signed quantities ranging from 2^{-31} to 2^{+31} can be passed from one component to another component in the BFPDDA.

In the BFPDDA floating point arithmetic is introduced into the conventional DDA structure in place of the normal fixed point arithmetic. The floating point arithmetic transforms the conventional DDA in many ways without losing its basic structure. The altered structure, the mathematical algorithms, and the operation of the BFPDDA are presented in the following sections.

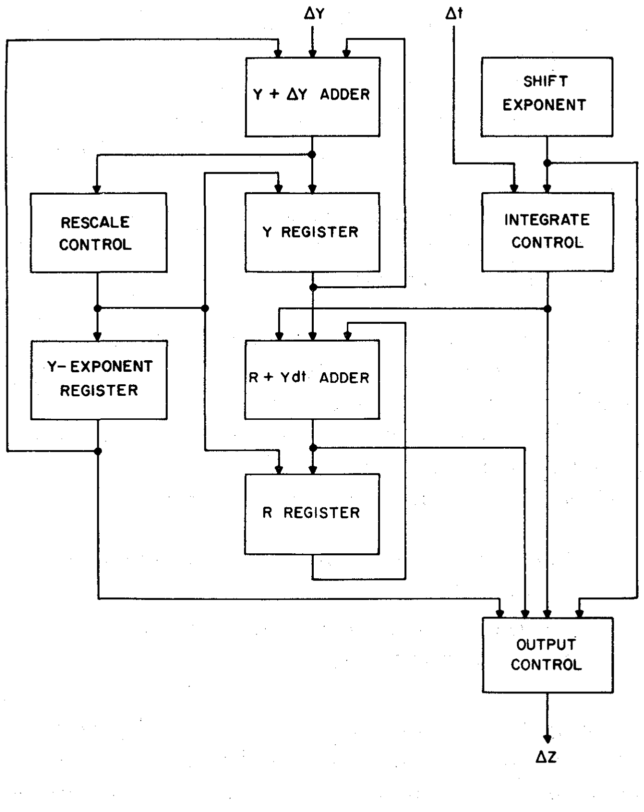


Figure 2—BFPDDA integrator block diagram

THE BFPDDA INTEGRATOR STRUCTURE

The BFPDDA should operate at the fastest speed possible in order to effectively replace the analog computer; therefore, the integrators should operate in parallel. Each integrator with its own adders and control units is assumed to be on an integrated circuit chip. Figure 2 is a block diagram of a proposed BFPDDA integrator showing its basic units and the lines of communication among these units. Note that the four units directly under the ΔY input are the same as the units in a DDA.

Let the current value of the integrand Y be represented by

$$Y = \pm .yyyy * 2^{k}$$

where yyy is the mantissa and k is the characteristic. Similarly, the residue R is represented by

$$R = \pm .rrrr * 2^{k+j}$$

where $|j|$ is the number of positions the Y register is shifted right so its bits are aligned with the bits of the R register. The value of j is negative so that $R < Y$. Using these definitions, general descriptions of each of the units making up the integrator are briefly given as follows.

$Y + \Delta Y$ Adder—This adder is used to increment the value of the integrand.

Y Register—The Y register contains the mantissa of the integrand.

$R + Y\Delta t$ Adder—This adder performs the integration.

R Register—The R register contains the mantissa of the residue.

Y-exponent Register—The Y-exponent is the characteristic of the integrand.

Rescale Control—The rescale control normalizes the integrand.

Shift Exponent—The shift exponent is the number of places which the R register is shifted left in order to be aligned with the Y register.

Integrate Control—This unit controls the information flow during each iteration.

Output Control—This unit calculates the output increment.

The Y and R registers contain mantissas of floating point numbers in binary coded form. In this paper the left most bit of the register is assumed to be the sign bit. The radix point is assumed to lie between the sign bit and the second bit of the register. Thus, the high order significant bit of the Y and R registers is the second bit. Thus, the $R + Y\Delta t$ adder is a simple integer adder.

Similarly, the Y-exponent and shift exponent registers contain exponents in a binary coded form. The Y-

exponent register contains the number of shift positions the Y register must be shifted to the left for the radix point to be properly positioned. The shift exponent is the number of places the R register is shifted from the Y register.

THE MATHEMATICAL ALGORITHMS FOR THE BFPDDA

The implementation of floating point arithmetic in the BFPDDA slightly alter the integration calculations used in the conventional DDA. The change in number representation requires an additional calculation to determine the output exponent. Finally, in order to make effective use of the dynamic scaling capabilities of a DDA with floating point arithmetic, algorithms for rescaling are included.

The integration phase of each iteration realizes the transfer function

$$R+Y\Delta t \rightarrow \Delta Z+R.$$

The Y and R values are represented by

$$Y=\pm .yyyy*2^{k}$$

and

$$R=\pm .rrrr*2^{k+j}$$

where $yyyy$ and $rrrr$ are the contents of the Y and R registers respectively. The Y-exponent register contains k and the shift exponent register contains j. Let

$$\Delta t=\pm 1.0*2^{i}$$

where $i \leq j$. Then the integration phase is described in Algorithms I and II, where the carry flag is a simple set and reset flip–flop.

Algorithm I.—Integration

```
1. Shift the $Y$ register $j-i$ positions to the right.
2. Add the shifted $Y$ register to the $R$ register.
3. If the $R$ register does not overflow, reset the carry flag;
   Otherwise,
   a. If the $R$ register is positive,
      i. Decrement the $R$ register by 1.0.
      ii. Set the sign bit associated with $\Delta Z$ to positive.
      iii. Set the carry flag.
   b. If the $R$ register is negative,
      i. Increment the $R$ register by 1.0.
      ii. Set the sign bit associated with $\Delta Z$ to negative.
      iii. Set the carry flag.
```

Noticing that requiring $i \leq j$ is a very practical restriction in the BFPDDA structure. The Y register is considered to be shifted $|j|$ positions left in order to be aligned with the R register. Therefore, if the step size of the independent variable is not at least as small as 2^{j}, a multiple bit overflow, which the DDA is not prepared to handle, could occur.

Algorthim II.—Output calculation

```
1. If the carry flag is set, transmit $k+j$ as $\Delta j$ along with the sign bit.
2. If the carry flag is reset, transmit no output.
```

During each iteration, the integrand must be updated so that it is as accurate as possible. The incrementation phase performs the transfer function

$$Y+\Delta Y \rightarrow Y.$$

The value of the integrand Y is the same as previously defined. The value of ΔY is of the form

$$\Delta Y=\pm 1.0*2^{m}$$

where m is the exponent being received on the ΔY input lines along with the correct sign. The procedure used for the incrementation of the integrand is now given in Algorithm III.

Algorithm III.—Incrementation of the integrand

```
1. If $k \leq m$, invoke Algorithm IV;
   Otherwise,
   a. Shift $\pm 1.0$ right $k-m-l$ positions.
   b. Add the shifted value to the $Y$ register.
   c. If the $Y$ register overflows, invoke Algorithm V.
   d. If the magnitude of the $Y$ register is less than one-half without considering the $Y$-exponent, invoke Algorithm VI.
```

An overflow condition occurs when the magnitude of ΔY is larger than the magnitude of Y. Since this means that the change in Y is larger than Y itself, the Y value is considered to be negligible. In this case the integrator is reset as defined in Algorithm IV.

Algorithm IV.—Resetting the integrand

```
1. Set the $Y$ register to $\pm 0.5$ depending on the sign of $\Delta Y$.
2. Set the $Y$-exponent register to $m+1$.
3. Set the $R$ register to zero.
```

During the incrementation phase of each iteration, the Y register is treated as a fixed point value. When the Y register overflows, a rescaling of the integrator is performed as described in Algorithm V.

Algorithm V.—Rescaling for an increasing integrand

1. Shift the Y register one position to the right.
2. Shift the R register one position to the right.
3. Increment the Y-exponent register by one.

Just as the value of the integrand may get larger in magnitude, it may also get smaller. By keeping the fractional value of the integrand normalized in the Y register, output overflows will tend to be smaller but will occur more frequently. The procedure for rescaling for a decreasing integrand is shown in Algorithm VI. Notice that this algorithm does not alter the R register. Many tests seemed to indicate that ignoring the R register gave the best results.

Algorithm VI.—Rescaling for a decreasing integrand

1. Shift the Y register one position to the left.
2. Decrement the Y-exponent register by one.

THE OPERATION OF A BFPDDA

As with the conventional DDA, the basic cycle of the BFPDDA is a two phase iteration. The integration and incrementation phases are outlined in Table I.

In phase I the integration is performed. While the Y register is being shifted and added to the R register, the output value is calculated. Then if an overflow occurs, the calculated output value is transmitted, otherwise, no output is transmitted and the calculated output value is ignored.

In phase II the integrand is incremented. An overflow in the addition causes the incremented value to be stored one position to the right and Y-exponent register to be incremented by one. An increment that is larger than the current value of the integrand causes the Y, Y-exponent, and R registers to be reset. On the other hand, if the mantissa is small enough, the result is stored one position to the left in the Y register and the Y-exponent is decremented by one. In this way the integrand remains normalized.

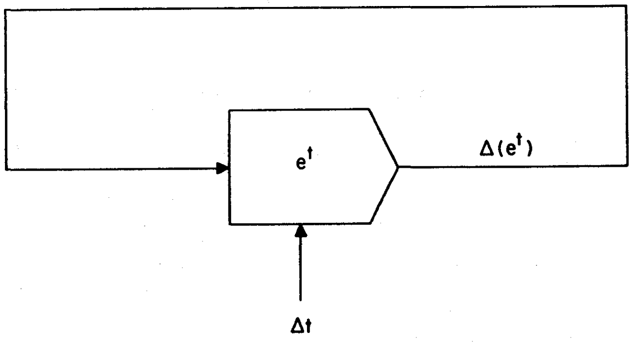


Figure 3—DDA program for e^t

TABLE I—BFPDDA Operational Iteration

I. Integration Phase	II. Incrementation Phase
$R+Y\Delta t\rightarrow\Delta Z+R$	$Y+\Delta Y\rightarrow Y$
1. Integration	1. Incrementation
2. Output Calculation	2. Rescaling

THE ELIMINATION OF MAGNITUDE SCALING

One of the major drawbacks of the ordinary DDA is the fixed point arithmetic it employs. Each integrator must be magnitude scaled in order that the integrand register doesn't overflow during the problem solution causing the program to abort. On the other hand, if the maximum value is overestimated by a significant amount, the error being delayed in the R register of the integrator is much larger, causing the problem error to be increased.

Consider the DDA solution of $\dot{y}-y=0$. The program for this equation is very simple as is shown in Figure 3. Despite the simple program, e^t is one of the more difficult solutions for the DDA to calculate. The exponential represents a continuously increasing function with a large variation in magnitude. Since the accumulated error will never be canceled, the magnitude of the error increases as the value of the independent variable t increases. Also, the initial value of the exponential function is its minimum value which results in very large initial errors since the integrator must be scaled for the values of increasing magnitude of the function.

Table II illustrates the effect of magnitude scaling on accuracy. When the integrator was scaled for a maximum value of 2^3, the error in calculating e^2 with a step size of 2^{-8} was approximately equal to the error in calculating e^2 with a step size of 2^{-13} in an integrator scaled for a maximum value of 2^8. Although e^t may be the extreme case, the exponential demonstrates the loss of accuracy in DDA problem solutions with variables that have a large variation in their magnitude.

The standard DDA has another scaling problem which the exponential exhibits. When the DDA is programmed, the step size must be fixed since the shift between the Y and R registers of the DDA is directly related to the step size. Thus the output line represents a fixed quantity and each output pulse is one unit of that quantity. If the step size is then changed, the out-

TABLE II—Comparison of Maximum Errors in Calculating $e^2=7.3891$ with a DDA Using Varying Magnitude Scalings

Step Size	Maximum Integrator Value $2^3=8$	$2^8=256$
2^{-8}	0.1553	3.3891
2^{-9}	0.0803	1.8891
2^{-10}	0.0406	1.0103
2^{-11}	0.0201	0.5141
2^{-12}	0.0102	0.2623
2^{-13}	0.0051	0.1309
2^{-14}	0.0026	0.0657

put quantity also changes. The new output quantity then must be reprogrammed at each point where it is used as an input to another component.

In calculating the exponential the integrator is magnitude scaled. Suppose the maximum value is set at 2^3 and a step size of 2^{-10} is selected, then each overflow or output bit represents 2^{-7}. The value of e^0 is loaded as the initial condition and e^2 is calculated. According to Table II an error of 0.0392 has occurred. If this error is determined to be too large, a smaller interval must be chosen. Choosing an interval of 2^{-12} makes each output bit represent 2^{-9}. Each increment of Y is then one-fourth of the previous increment size and the integrator must be reconnected to allow for these smaller incremental values.

Fixed point arithmetic in the regular DDA also causes scaling problems for integrators with more than one input incrementing its integrand value. Since the DDA is incremented by receiving a pulse, the increment value is fixed within the integrator being incremented. Therefore, when the integrator receives increments from two or more components, the inputs must represent the same value.

The magnitude scaling problem does not exist in the BFPDDA. As the integrand values become larger the integrator rescales upward. Similarly, the integrator rescales downward as the values become smaller in magnitude. The user does not have to estimate the maximum values or even know much about the relationships among the integrators since each integrator functions independently of the other integrators without regard to the integrand magnitude.

Since the BFPDDA integrators rescale themselves, the accuracy of the BFPDDA is better than that of a regular DDA. When the integrand magnitude becomes smaller, the integrator rescales downward causing the fractional error of the integral in the R register to be reduced. A smaller overflow then occurs in the integration; however, the overflow occurs much sooner and introduces much less delay error than in the standard DDA. Thus the BFPDDA causes more, but smaller overflows than the ordinary DDA in solving most integrals which leads to more accurate solutions.

Two solutions of e^t were calculated using the BFPDDA. After each iteration the calculated value was compared with the actual value and the error determined. The maximum errors in calculating e^2 and e^5 for varying interval sizes is shown in Table III. The same values were also calculated with a regular DDA and also appear. When the magnitude scaling was done for e^2, only a three bit binary shift was necessary and the DDA was practically the same as the BFPDDA. However, when an allowance of eight bits was made for e^5 in the DDA and two and one-half times as many iterations were performed, the BFPDDA was over ten times as accurate as the DDA. The BFPDDA could then solve e^5 ten times as fast as the DDA for the same accuracy, since speed and accuracy are inversely proportional.

The values appearing in Table III indicate that the BFPDDA is more accurate because of its rescaling techniques and floating point arithmetic. Over the shorter solution of e^2 the accumulative error wasn't too critical. When a slightly longer problem was solved with a larger range in integrand magnitudes, the accumulative error in the DDA greatly impaired its advantages but only minimally effected the BFPDDA.

Other factors not appearing in Table III also make the BFPDDA preferable. In changing from one interval size to another, the shift exponent register was merely reloaded with the exponent of the step size chosen for maximum accuracy. Otherwise, there was no reconnection for the new problem solution. The values taken from the BFPDDA were directly readable. There was no multiplication by a scaling factor necessary to put the calculated result in a form relative to that of the unscaled, original equation.

TABLE III—Comparison of BFPDDA and DDA in Accuracy Solving e^t

Maximum Error in Calculating $e^2=7.3891$		Step Size	Maximum Error in Calculating $e^5=148.41$	
DDA	BFPDDA		DDA	BFPDDA
0.1553	0.1524	2^{-8}	65.413	5.543
0.0803	0.0790	2^{-9}	35.624	2.835
0.0406	0.0392	2^{-10}	18.413	1.413
0.0201	0.0198	2^{-11}	9.393	0.716
0.0102	0.0099	2^{-12}	4.726	0.359
0.0051	0.0050	2^{-13}	2.367	0.180
0.0026	0.0025	2^{-14}	1.187	0.091

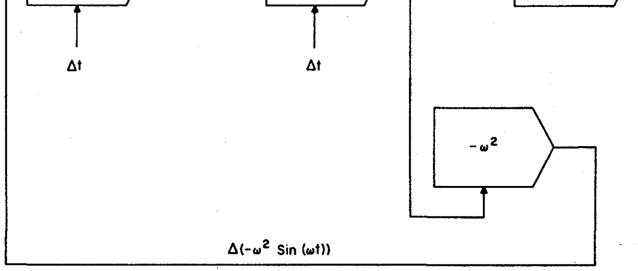
Another advantage of the BFPDDA appears when second or higher order equations are solved. An integrator receives its integrand increments in the form of an exponent instead of a pulse. Since the magnitude of the increment is passed between the integrators, the integrator may receive increments of different magnitudes during the problem solution. Thus, the integrator may have two or more increment inputs without having the multiple inputs scaled to represent the same magnitude.

In the BFPDDA magnitude scaling has been eliminated. Each integrator dynamically rescales itself independently as the magnitude of its integrand varies. The accuracy in the BFPDDA is not dependent on the estimated maximum value used in the magnitude scaling. With the BFPDDA the independent variable step size may be changed easily with no reconnection necessary. Finally, an integrator receiving an increment from more than one component does not require all of the incremental values to be of equal magnitude.

BFPDDA SOLUTION OF THE HARMONIC EQUATION

The BFPDDA program for solving the harmonic equation, $\ddot{x}+\omega^2x=0$ is shown in Figure 4. The result of the solution is obtained at any given time during the solution by reading the current value contained in the sin ωt integrator, which in this problem is simply used as a summer. In the results shown in this section, sin ωt was read after each iteration and the error calculated at each point.

The BFPDDA can solve the same problem with many different parameters without being rescaled or reprogrammed. Even large variations in the magnitudes of the parameters have almost no effect. Table IV displays the maximum error that occurred during the one cycle solution of a sine wave. The frequencies were generated randomly from successive ranges bounded by powers of two.



$-\omega^2$ Sin (ωt)
$\Delta(\omega \cos(\omega t))$
ω Cos(ωt)
Δ (Sin (ωt))
Sin (ωt)
Δt
Δt
$-\omega^2$
$\Delta(-\omega^2$ Sin $(\omega t))$

Figure 4—Interconnection of BFPDDA integrators to solve harmonic equation

TABLE IV—Maximum Errors in One Cycle Sine Wave Solutions

ω^2	Step Size 2^{-10}	2^{-12}	2^{-14}
0.4437	0.00662	0.00165	0.00040
0.5653	0.00881	0.00217	0.00055
1.8120	0.01093	0.00271	0.00064
3.3890	0.01328	0.00361	0.00081

In the BFPDDA all the values shown in Table IV were calculated without reprogramming or rescaling. In the standard DDA a change to a smaller step size would necessitate reconnection so that the proper values were represented. Similarly, the DDA has to be rescaled each time the step size changes or the magnitude of the frequency is significantly altered. In the case of the BFPDDA the step size is changed by simply reloading the shift exponent registers and setting a smaller exponent on the independent variable input lines. A new frequency is simply reloaded in the Y register of the integrator being used for constant multiplication and the initial condition reset on the $\omega \cos(\omega t)$ integrator in order to solve the harmonic equation for a new ω^2 value.

The maximum errors increase as the frequency increases since one cycle of the sine wave is broken into fewer intervals. Thus, as higher frequency sine waves are generated, smaller intervals may be necessary in order to maintain accuracy. The smaller intervals will not increase the overall problem time, however, since the range of one cycle is much smaller.

The harmonic equation was also used for comparing the accuracy of the BFPDDA against the accuracy of the DDA. For this test $\omega^2=1$ and the integrator containing ω^2 was replaced by an inverter on the sign transfer bit. The results which appear in Table V show that the BFPDDA is twice as accurate as the DDA.

CONCLUSIONS

The emphasis in designing the BFPDDA has been on improving user convenience.[9] By reconstructing a standard DDA to use floating point arithmetic and to transfer exponents between its components, an easily programmable device requiring no magnitude scaling resulted. Moreover, the floating point arithmetic proved the BFPDDA to be more accurate than the

TABLE V—Maximum Errors in a One Cycle Solution of the Harmonic Equation with $\omega = 1$

Step Size	DDA	BFPDDA
2^{-8}	0.03209	0.01605
2^{-9}	0.01477	0.00797
2^{-10}	0.00709	0.00406

ordinary DDA. The accuracy improvement becomes more significant as the variation in magnitude of the problem variables increases. The BFPDDA also allows for alterations in the rate of integration during a problem solution since no new scaling is necessary.

Considering current technology and the BFPDDA, the hybrid computing system could be headed for new horizons. The BFPDDA with all main advantages of digital computation in an analog environment will be an excellent special purpose differential equation solver on-line with a time-shared digital computer. Dynamical systems with unknown solutions can quickly be solved since the solution will not depend on estimated maximum parameter values. Being digital the whole field of automatic patching by program can make the BFPDDA easier to use. Making the DDA floating point and greatly increasing user convenience at only a very slight cost increase should make hybrid computation very popular.

REFERENCES

1 M W GOLDMAN
Design of a high speed DDA
AFIPS Conference Proceedings Fall Joint Computer Conference pp 929–949 1965

2 R B McGHEE R N NILSEN
The extended resolution digital differential analyzer: a new computing structure for solving differential equations
IEEE Trans on Computers Vol C-19 pp 1-9 January 1970

3 T C BARTEE J B LEWIS
A digital system for on-line studies of dynamical systems
AFIPS Conference Proceedings Spring Joint Computer Conference pp 105–111 1966

4 M W HOYT W T LEC O A REICHARDT
The parallel digital differential analyzer and its application as a hybrid computing system element
Simulation Vol 4 pp 104–113 February 1965

5 T C BARTEE I L LEBOW I S REED
Theory and design of digital machines
McGraw-Hill New York pp 252–265 1962

6 H D HUSKEY G N KORN
Computer handbook
McGraw-Hill New York Chapter 3 pp 14–74 1962

7 A GILL
Systematic scaling for digital differential analyzers
IRE Trans on Electronic Computers Vol EC-8 pp 486-489 December 1959

8 H K KNUDSEN
The scaling of digital differential analyzers
IEEE Trans on Electronic Computers Vol EC-14 pp 583-590 August 1965

9 J L ELSHOFF
The binary floating point digital differential analyzer
PhD dissertation The Pennsylvania State University University Park Pennsylvania September 1970