Automated Discovery of Numerical Approximation Formulae via Genetic Programming

Matthew Streeter

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Lee A. Becker

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

This paper describes the use of genetic programming to automate the discovery of numerical approximation formulae. The authors present results involving rediscovery of known approximations for Harmonic numbers and discovery of rational polynomial approximations for functions of one or more variables, the latter of which are compared to Padé approximations obtained through a symbolic mathematics package. For functions of a single variable, it is shown that evolved solutions can be considered superior to Padé approximations, which represent a powerful technique from numerical analysis, given certain tradeoffs between approximation cost and accuracy, while for functions of more than one variable, we are able to evolve rational polynomial approximations where no Padé approximation can be computed. Further, it is shown that evolved approximations can be refined through the evolution of approximations to their error function. Based on these results, we consider genetic programming to be a powerful and effective technique for the automated discovery of numerical approximation formulae.

1 INTRODUCTION

1.1 MOTIVATIONS

Numerical approximation formulae are useful in two primary areas: firstly, approximation formulae are used in industrial applications in a wide variety of domains to reduce the amount of time required to compute a function to a certain degree of accuracy (Burden and Faires 1997), and secondly, approximations are used to facilitate the simplification and transformation of expressions in formal mathematics. The discovery of approximations used for the latter purpose generally requires human intuition and insight, while approximations used for the former purpose tend to be polynomials or rational polynomials obtained by a technique from numerical analysis such as Padé approximants (Baker 1975; Bender and Orszag 1978) or Taylor series. Genetic programming (Koza 1992) provides a unified approach to the discovery of approximation formulae which, in addition to having the obvious benefit of automation, provides a power and flexibility that potentially allows for the evolution of approximations superior to those obtained using existing techniques from numerical analysis.

1.2 EVALUATING APPROXIMATIONS

In formal mathematics, the utility or value of a particular approximation formula is difficult to analytically define, and depends perhaps on its syntactic simplicity, as well as the commonality or importance of the function it approximates. In industrial applications, in contrast, the value of an approximation is uniquely a function of the computational cost involved in calculating the approximation and the approximation's associated error. In the context of a specific domain, one can imagine a utility function which assigns value to an approximation based on its error and cost. We define a reasonable utility function to be one which always assigns lower (better) scores to an approximation a_1 which is *unequivocally superior* to an approximation a_2, where a_1 is defined to be unequivocally superior to a_2 iff. neither its cost nor error is greater than that of a_2, and at least one of these two quantities is lower than the corresponding quantity of a_2. Given a set of approximations for a given function (obtained through any number of approximation techniques), one is potentially interested in any approximation which is not *unequivocally inferior* (defined in the natural way) to any other approximation in the set. In the terminology of multi-objective optimization, this subset is referred to as a Pareto front (Goldberg 1989). Thus, the Pareto front contains the set of approximations which could be considered to be the most valuable under some reasonable utility function.

1.3 RELATED WORK

The problem of function approximation is closely related to the problem of function identification or symbolic regression, which has been extensively studied by

numerous sources including (Koza 1992; Andre and Koza 1996; Chellapilla 1997; Luke and Spector 1997; Nordin 1997; Ryan, Collins, and O'Neill 1998). Approximation of specific functions has been performed by Keane, Koza, and Rice (1993), who use genetic programming to find an approximation to the impulse response function for a linear time-invariant system, and by Blickle and Thiele (1995), who derive three analytic approximation formulae for functions concerning performance of various selection schemes in genetic programming. Regarding general techniques for the approximation of arbitrary functions, Moustafa, De Jong, and Wegman (1999) use a genetic algorithm to evolve locations of mesh points for Lagrange interpolating polynomials.

2 EVOLVING NUMERICAL APPROXIMATION FORMULAE USING GENETIC PROGRAMMING

All experiments reported in this paper make use of the standard genetic programming paradigm as described by Koza (1992). Our task is to take a function in symbolic form (presented to the system as a set of training points) and return a (possibly singleton) set of expressions in symbolic form which approximate the function to various degrees of accuracy. The authors see two essential methods of applying genetic programming to this task: either by limiting the available function set in such a way that the search space contains only approximations to the target function, rather than exact solutions, or by in some way incorporating the computational cost of an expression into the fitness function, so that the evolutionary process is guided toward simpler expressions which presumably will only be able to approximate the data. Only the former approach is considered here.

The system used for the experiments described in this paper was designed to be functionally equivalent to that described by Koza (1992) with a few minor modifications. Firstly, the evolution of approximation formulae requires the cost of each approximation to be computed. We accomplish this by assigning a raw cost to each function in the function set, and taking the cost of an approximation to be the sum of the functional costs for each function node in its expression tree whose set of descendent nodes contains at least one input variable. For all experiments reported in this paper, the function costs were somewhat arbitrarily set to 1 for the functions `/`, `*`, and `RCP` (the reciprocal function), 0.1 for the functions + and −, and 10 for any more complex function such as `EXP`, `COS`, or `RLOG`.

Secondly, this system uses a slightly modified version of the standard adjusted fitness formula 1/(1+[error]) which attempts to maintain selection pressure when error values are small. We note that although an approximation which attains an error of 0.1 is twice as accurate as one with an error of 0.2, the standard formula will assign it an adjusted fitness which is just over 9% greater. We attempt to avoid this problem by introducing an *error multiplier*, so that the adjusted fitness formula becomes 1/(1+[error multiplier][error]). For one experiment described in this paper, the error multiplier was set to 1000. In the given example, this causes the approximation with an accuracy of 0.1 to have a fitness which is nearly twice (~1.99 times) that of the approximation whose accuracy is 0.2, which is more appropriate.

Finally, rather than simply reporting the best (i.e. most accurate) approximation evolved in each of a number of runs, we report the Pareto front for the union of the population histories of each independent run, computed iteratively and updated at every generation. Thus, this system returns the subset of approximations which are potentially best (under some reasonable utility function) from the set of all approximations evolved in the course of all independent runs.

The integrity of the system used in these experiments, which was written by the authors in C++, was verified by reproducing the experiment for symbolic regression of $f(x) = x^4 + x^3 + x^2 + x$ as reported by Koza (1992).

3 REDISCOVERY OF HARMONIC NUMBER APPROXIMATIONS

One commonly used quantity in mathematics is the Harmonic number, defined as:

$$H_n \equiv \sum_{i=1}^{n} 1/i$$

This series can be approximated using the asymptotic expansion (Gonnet 1984):

$$H_n = \gamma + \ln(n) + 1/(2n) - 1/(12n^2) + 1/(120n^4) - \ldots$$

where γ is Euler's constant ($\gamma \approx 0.57722$).

Using the system described in the previous section, and the function set `{+,*,RCP,RLOG,SQRT,COS}`, the authors attempted to rediscover some of the terms of this asymptotic expansion. Here `RLOG` is the protected logarithm function (which returns 0 for a non-positive argument) and `RCP` is a protected reciprocal function which returns the reciprocal of its argument if the argument is non-zero, or 0 otherwise. `SQRT` and `COS` are included as extraneous functions.

All parameter settings used in this experiment are the same as those presented as the defaults in John Koza's first genetic programming book (Koza 1992), including a population size of 500 and generation limit of 51. The first 50 Harmonic numbers (i.e. H_n for 1<=n<=50) were used as training data. 50 independent runs were executed, producing a single set of candidate approximations. Error was calculated as the sum of absolute error for each training instance. The error multiplier set to 1 for this experiment (e.g. effectively not used).

The set of evolved approximations returned by the genetic programming system (which represent the Pareto front for

the population histories of all independent runs) is given in Table 1. For the purpose of analysis, each approximation was simplified using the Maple symbolic mathematics package; for the sake of brevity, only the simplified expressions (rather than the full LISP expressions) are given in this table.[1]

Table 1. Evolved Harmonic Number Approximations.

SIMPLIFIED EXPRESSION			
ERROR	**COST**	**RUN**	**GENERATION**
1. ln(x)+.5766598187+1/(sqrt(ln(x)+.5766598187 +1/(1/x+2*x+.6426220121)+x^2)+x)			
0.0215204	39.1	22	32
2. ln(x)+.5766598187+1/(2*x+1/(1.219281831 + ln(1/(ln(x)+.5766598187))+x))			
0.0229032	35.8	22	35
3. ln(x)+.5766598187+1/(2*x+1/(1.285244024 + ln(1.734124639+2*x)))			
0.0264468	26.9	22	37
4. ln(x)+.5766598187+1/(2*x+1/(2.584025920 + ln(x)+1/(3.007188263+x)))			
0.0278816	25.9	22	49
5. ln(x)+.5766598187+1/(2*x+1/(.5766598187 + 1/x+x))			
0.0286254	15.7	22	36
6. ln(x)+.5766598187+1/(2*x+.3592711879)			
0.0293595	13.4	22	37
7. ln(x)+.5766598187+1/(2*x+.3497550998)			
0.0297425	11.4	22	42
8. ln(x+.5022291180)+.5779513609			
0.0546846	10.3	40	28
9. ln(x+.4890238595)+.5779513609			
0.0653603	10.2	40	21
10. 0.5965804779+ln(x)			
1.44089	10.1	49	49
11. 3.953265289-4.348430001/x			
20.2786	2.2	3	1
12. 3.815981083			
31.0297	0	10	4

[1] Note that since the cost and error values given in Table 1 were calculated by the genetic programming system (using the unsimplified versions of the approximations), the cost values are not necessarily the same as those which would be obtained by manually evaluating the simplified Maple expressions.

An analysis of this set of candidate solutions follows. For comparison, Table 2 presents the error values associated with the asymptotic expansion when carried to between 1 and 4 terms.

Table 2. Accuracy of Asymptotic Expansion

TERMS	EXPRESSION	ERROR
1	0.57722	150.559
2	0.57722 + ln(n)	2.12094
3	0.57722 + ln(n) + 1/(2n)	0.128663
4	0.57722 + ln(n) + 1/(2n) - 1/(12n^2)	0.00683926

Candidate approximation 12, the cheapest approximation in the set, is simply a constant, while candidate approximation 11 is a simple rational polynomial. Candidate approximation 10 represents a variation on the first two terms of the asymptotic expansion, with a slightly perturbed version of Euler's constant which gives greater accuracy on the 50 supplied training instances. Candidate solutions 8 and 9 represent slightly more costly variations on the first two terms of the asymptotic expansion which provide increased accuracy over the training data. Similarly, candidate solutions 6 and 7 are slight variations on the first three terms of the asymptotic expansion, tweaked as it were to give greater accuracy on the 50 training points. Candidate solutions 2-5 can be regarded as more complicated variations on the first three terms of the asymptotic expansion, each giving a slight increase in accuracy at the cost of a slightly more complex computation. Candidate solution 1 represents a unique and unexpected approximation which has the greatest accuracy of all evolved approximations, though it is unequivocally inferior to the first four terms of the asymptotic expansion has presented in Table 2.

Candidate approximations 1-7 all make use of the constant 0.5766598187 as an approximation to Euler's constant, which was evolved using the LISP expression:

```
(RCP(SQRT(* 4.67956 RLOG(1.90146))))
```

This approximation is accurate to two decimal places. Candidate approximations 8 and 9 make use of the slightly less accurate approximation of 0.5779513609, evolved using the LISP expression:

```
(COS(LN 2.59758))
```

Note that in this experiment, pure error-driven evolution has produced a rich set of candidate approximations exhibiting various trade-offs between accuracy and cost. Also note that with the exception of the first candidate approximation, which uses the `SQRT` function, the `SQRT` and `COS` functions were used only in the creation of constants, so that these extraneous functions did not provide a significant obstacle to the evolution of the desired approximations. Thus, this experiment represents

a partial rediscovery of the first three terms of the asymptotic expansion for H_n.

4 DISCOVERY OF RATIONAL POLYNOMIAL APPROXIMATIONS FOR KNOWN FUNCTIONS

4.1 INTRODUCTION

By limiting the set of available functions to the arithmetic function set {*, +, /, −}, it is possible to evolve rational polynomial approximations to functions, where a rational polynomial is defined as the ratio of two polynomial expressions. Since approximations evolved with the specified function set use only arithmetic operators, they can easily be converted to rational polynomial form by hand, or by using a symbolic mathematics package such as Maple. Approximations evolved in this manner can be compared to approximations obtained through other techniques such as Padé approximations by comparing their Pareto fronts. In the section, we present the results of such a comparison for three common mathematical functions: the natural logarithm ln(x), the square root sqrt(x), and the hyperbolic arcsine arcsinh(x), approximated over the intervals [1,100], [0,100], and [0,100], respectively. The functions were selected to be common, aperiodic functions whose calculation was sufficiently complex to warrant the use of approximation. The intervals were chosen to be relatively large due to the fact that Padé approximations are weaker over larger intervals, and we wished to construct examples for which the genetic technique might be most applicable.

4.2 COMPARISON WITH PADÉ APPROXIMATIONS

The Padé approximation technique is parameterized by the value about which the approximation is centered, the degree of the numerator in the rational polynomial approximation, and the degree of the denominator. Using the Maple symbolic mathematics package, we calculated all Padé approximations whose numerator and denominator had a degree of 20 or less, determined their associated error and cost, and calculated their (collective) Pareto front for each of the three functions being approximated. The center of approximation was taken as the leftmost point on the interval for all functions except the square root, whose center was taken as x=1 since the necessary derivatives of sqrt(x) are not defined for x=0. Error was calculated using a Riemann integral with 1000 points. For simplicity, the cost of Padé approximations was taken only as the minimum number of multiplications/divisions required to compute the rational polynomial, as calculated by a separate Maple procedure.

The Maple procedure written to compute the cost of an approximation operated by first putting the approximation in continued-fraction form (known to minimize the number of necessary multiplications/divisions), counting the number of multiplications/divisions required to compute the approximation in this form, and then subtracting for redundant multiplications. As an example of a redundant multiplication, the function $f(x)=x^2+x^3$ when computed literally requires 3 multiplications (1 for x^2, 2 for x^3), but need be computed using only 2, since in the course of computing x^3 one naturally computes x^2.

For consistency, the candidate approximations evolved through the genetic programming technique were also evaluated (subsequent to evolution) using the Reimann integral and Maple cost procedure, and the Pareto front for this set of approximations was recomputed using the new cost and error values. Finally, it should be noted that a Padé approximation with denominator of degree zero is identical to the Taylor series whose degree is that of the numerator, so that the Pareto fronts reported here effectively represent the potentially best (under some reasonable utility function) members of a set of 20 Taylor series and 380 uniquely Padé approximations.

4.3 RESULTS

All experiments involving rational polynomial approximations were performed using the same settings as described in the previous section, but with a generation limit of 101 (we have found that accurate rational polynomial approximations take a while to evolve). The / function listed in the function set was defined to be a protected division operator which returns the value 10^6 if division by zero is attempted. In analyzing evolved approximations via Maple, any approximation which performed division by zero was discarded. To reduce the execution time of these experiments, we employed the technique suggested as a possible optimization by Koza (1990) of using only a subset of the available training instances to evaluate individuals at each generation. In our experiments, the subset is chosen at random for the initial generation, and selected as the subset of examples on which the previous best-of-generation individual performed the worst for all subsequent generations. The subset is assigned a fixed size for all generations; for all experiments reported in this section, the subset size was 25. Training data consisted of 100 points, uniformly spaced over the interval of approximation. Each of the three experiments reported was completed in approximately 4-5 hours on a 600 MHz Pentium III system.

Figures 1-3 present the Pareto fronts for Padé approximations and for genetically evolved approximations of the functions ln(x), sqrt(x), and arcsinh(x), respectively, evaluated over the intervals [1,100], [0,100], and [0,100], respectively. In each of these three figures, the dashed line connects points corresponding to Padé approximations, while the solid line connects points corresponding to genetically evolved approximations. All Padé approximations not accounted for in computing the Pareto front represented by the dashed line (i.e. all Padé approximation whose numerator or denominator has a degree larger than 20) must involve

at least 20 multiplications/divisions, if only to compute the various powers of x: x, x^2, x^3, . . . x^{21}. For this reason, a dashed horizontal line at *cost*=20 is drawn in each figure, so that the horizontal line, combined with the dashed lines representing the Pareto front for Padé approximations with numerator and denominator of degree at most 20, represents the best case Pareto front for all Padé approximations of any degree.

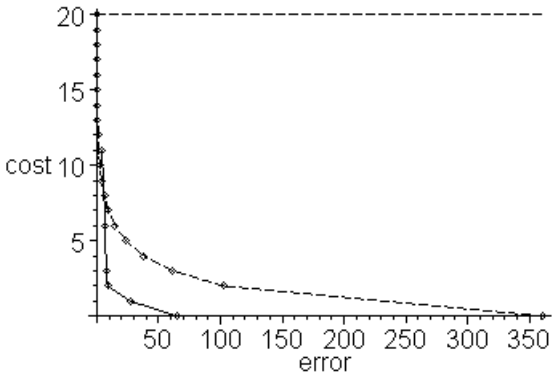

Figure 1: Pareto Fronts for Approximations of ln(x).

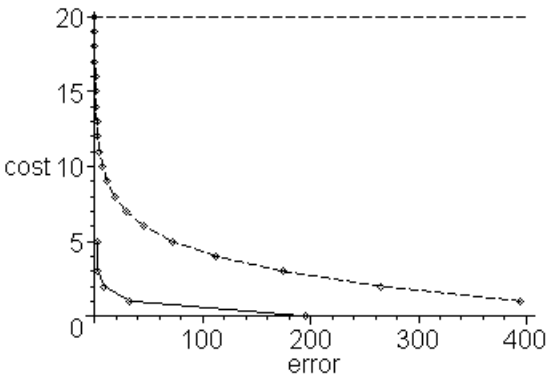

Figure 2: Pareto Fronts for Approximations of sqrt(x).

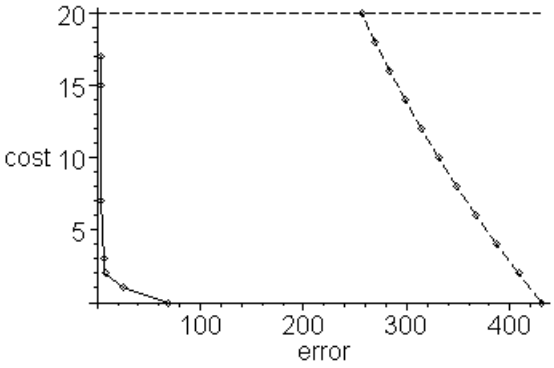

Figure 3: Pareto Fronts for Approximations of arcsinh(x).

For each experiment, we are interested in the genetically evolved approximations which lie to the interior of the Pareto fronts for Padé approximations, and thus are superior to Padé approximations given certain trade-offs between error and cost. Tables 3-5 list all such approximations for ln(x), sqrt(x), and arcsinh(x), respectively, along with their associated cost and error as calculated by the Maple cost procedure and by a Riemann integral, respectively. For ln(x), we are able to obtain 5 approximations which lie to the interior of the Pareto front for Padé approximations, for sqrt(x) we are also able to obtain 5 such approximations, and for arcsinh(x) we are able to obtain 7 approximations, all exhibiting various trade-offs between error and cost. As can be seen from Figure 3, arcsinh(x) proved to be a particularly difficult function for Padé approximations to model over the given interval.

Table 3: Evolved Approximations for ln(x).

EXPRESSION	COST	ERROR
(.0682089-2*x-x/(-.1218591501*x-3.842080570))/(-.385143144*x-4.6585)	6	6.798897089
1.426990291*x/(4.132660+.2760372098*x)	3	7.436110884
(4.205966*x-6.601615)/(x+12.85128201)+.694754	2	8.743267301
4.70397-29.12598131/(x+2.82952)	1	26.93968611
3.91812	0	64.55919780

Table 4: Evolved Approximations for sqrt(x).

EXPRESSION	COST	ERROR
x/(x/(4.78576+x/(9.17981+x/(15.39292+.04005697704*x)))+1.48335)	5	2.591348148
(x+.06288503787)/((x-9.04049)/(.05822627334*x+8.30072)+4.32524)+.795465	3	3.123452980
x/(5.5426193+.06559635887*x)+1.48335	2	8.935605674
.07262106112*x+3.172308452	1	32.95322345
7.011926	0	195.5193204

Table 5: Evolved Approximations for arcsinh(x).

EXPRESSION	COST	ERROR
1.86636*(1.277853316*x/((.3868816181*(-2.90216-x)/(-4.88586-x)+1.02145)*(-1.122792357-.3868816181*x))-.03522759767*(-1.122792357-.3868816181*x)*(x+4.86602)*(x-.269326)/(.0840785+x)+4.83551*x)/(9.684284+2.08151*x)	17	3.361399200
1.86636*(.07017092454*x^2/((2*x+4.86602)*(3.111694208+4.83551*x))-.03539134480*(.2502505059-.3868816181*x)*(x+4.86602)*(x-.269326)/(.0840785+x)+4.83551*x)/(9.684284+2.08151*x)	15	3.533969225
1.86636*(.0840785-.03522759767*(-1.122792357-.3868816181*x)*(x-.269326)+4.83551*x)/(9.684284+2.08151*x)	7	3.804858563
2.46147/(.4180284579-4.28068*1/(-2.299172064-.7261005920*x))	3	6.596080331
4.466119361*x/(18.01575130+x)+1.32282	2	7.581253733
3.30409+.02369172723*x	1	25.83927515
4.600931145	0	68.51916981

5 APPROXIMATING FUNCTIONS OF MORE THAN ONE VARIABLE

For some functions of more than one variable, it is possible to obtain a polynomial or rational polynomial approximations using techniques designed to approximate functions of a single variable; this can be done by nesting and combining approximations. For example, to obtain a rational polynomial approximation for the function $f(x,y)$=ln(x)*sin(y), one could compute a Padé approximation for ln(x) and a Padé approximation for cos(x) and multiply the two together. To compute a rational polynomial approximation for a more complex function such as $f(x,y)$=cos(ln(x)*sin(y)), one could again compute two Padé approximations and multiply them together, assign the result to an intermediate variable z, and compute a Padé approximation for cos(z). However, for any function of more than one variable that involves a non-arithmetic, non-unary operator whose set of operands contains at least two variables, there is no way to compute a polynomial or rational polynomial approximation using techniques designed to compute approximations for functions of a single variable. For the function $f(x)=x^y$, for example, there is no way to use Padé approximations or Taylor series to obtain an approximation, since the variables x and y are inextricably entwined by the exponentiation operator. In contrast, the genetic programming approach can be used on any function for which data points can be generated. To test the ability of genetic programming to evolve rational polynomial approximations for the type of function just described, an experiment was conducted to evolve approximations of the function $f(x)=x^y$ over the area $0<=x<=1$, $0<=y<=1$. Parameter settings were the same as described in the section on Harmonic numbers, including the generation limit of 51. Training data consisted of 100 (three dimensional) points chosen at random from the given rectangle. As in the previous section, a subset of 25 examples was used to evaluate the individuals of each generation.

The approximations returned by the genetic programming system were further evaluated through Maple. As in the previous section, a Maple procedure was used to calculate the minimum number of multiplications/divisions necessary to compute the approximation, while the error was evaluated using a double Riemann integral with 10000 points. The Pareto front for this set of approximations was then recomputed using the new cost and error values. The results of this evaluation are presented in Table 6.

Table 6: Evolved Approximations for x^y.

EXPRESSION	COST	ERROR
$x/(y\text{^}2+x-x*y\text{^}3)$	4	.03643611691
$x/(y\text{^}2+x-x*y\text{^}2)$	3	.04650160477
$x/(y+x-x*y)$	2	.04745973920
$x*y-y+.989868$	1	.05509570980
$x+.13336555$	0	.1401316648

The most accurate approximation evolved as a result of this experiment was $x/(y^2+x-xy^3)$. Figures 4 and 5 present graphs for the target surface $f(x)=x^y$ and for this approximation, respectively. Visually, the evolved surface is quite similar to the target function.

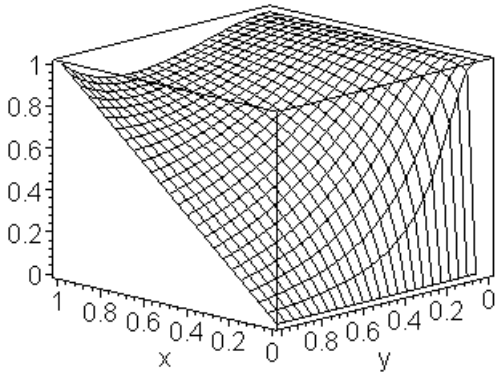

Figure 4: $f(x)=x^y$.

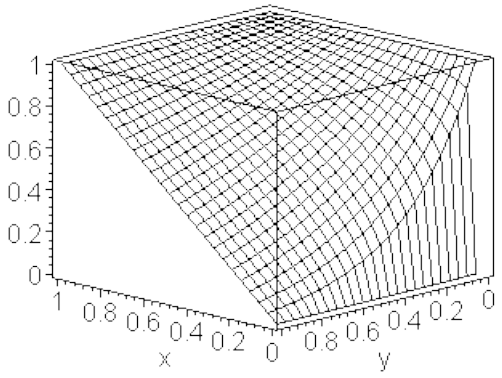

Figure 5: $x/(y^2+x-xy^3)$.

6 REFINING APPROXIMATIONS

It is possible to refine an approximation $a(x)$ by evolving an approximation ($a'(x)$) to its error function, then taking the refined approximation as $a(x)+a'(x)$. To test the practicality of this idea, we performed refinement of several evolved approximations to the function $\sin(x)$ over the interval $[0,\pi/2]$. Available space prohibits us from presenting the full results of this experiment. We note, however, that we are able to obtain 4 approximations in this manner which improve upon the Pareto front for our original experiment (prior to refinement), which contains a total of 7 approximations. The experiment was conducted using the same settings as in sections 4 and 5, but with an error multiplier of 1000. Refinement in this manner could be applied iteratively, to produce successively more accurate approximations. We have not investigated this possibility in any detail, but it is clear from our preliminary findings that the technique of refining approximations in this manner is indeed capable of producing significantly more accurate approximations.

In addition to refining evolved approximations using genetic programming, it is also possible to refine approximations generated through some other technique (such as Padé approximations) through genetic programming, or to refine approximations evolved via genetic programming through a technique from numerical analysis. Were the latter approach to prove effective, it could be incorporated on-the-fly in the evalution of individual approximations; one can imagine a rather different approach to the problem in which all evolving approximations are refined to a certain degree of accuracy by adding terms based on Padé approximations or Taylor series, and fitness is taken simply as the cost of the resulting expression. This provides for an interesting possible extension of the work reported in this paper.

7 FUTURE WORK

The work presented in this paper suggests a number of possible extensions. First, by adding if-then functions and appropriate relational operators such as less-than and greater-than to the function set, one could evolve piecewise rather than unconditional approximations to functions. Second, as suggested in the previous section, several extensions to this work based on the refinement of approximations could be attempted. Third, little attempt was made in this work to optimize parameters for the problem of finding rational polynomial approximations in general, and no parameter optimizations were made for specific functions being approximated, so that alteration of parameter settings represents a significant potential for improvement on the results presented in this paper. These results could also presumably be improved by using additional computational power and memory, and by employing a genetic programming system which allows for automatically defined functions (Koza 1994).

Perhaps the ideal application of this technique would be to perform the equivalent of conducting the Harmonic number experiment prior to 1734, the year that Leonhard Euler established the limiting relation

$$\lim n\to\infty H_n-\ln(n) \equiv \gamma$$

which defines Euler's constant (Eulero 1734). Such a result would represent "discovery" of an approximation formula in the truest sense, and would be a striking and exciting application of genetic programming.

8 CONCLUSIONS

This paper has shown that genetic programming is capable of rediscovering approximation formulae for Harmonic numbers, and of evolving rational polynomial approximations to functions which, under some reasonable utility functions, are superior to Padé approximations. For common mathematical functions of a single variable approximated over a relatively large interval, it has been shown that genetic programming can provide a set of rational polynomial approximations whose Pareto front lies in part to the interior of the Pareto front for Padé approximations to the same function. Though it has not been demonstrated explicitly in this paper, one would expect that genetic programming would also be able to expand upon the Pareto front for approximations to functions of more than one variable obtained by combining and nesting Padé approximations. Furthermore, for at least one function of more than one variable, genetic programming has been shown to provide a way to evolve rational polynomial approximations where the Padé approximation technique cannot be applied. Finally, we have presented results involving evolutionary refinement of evolved approximations. Based upon these results, the authors regard the genetic programming approach described in this paper as a powerful, flexible, and effective technique for the automated discovery of approximations to functions.

Acknowledgments

The authors wish to thank Prof. Micha Hofri of Worcester Polytechnic Institute for valuable advice and feedback received during the course of this project.

References

D. Andre and J. R. Koza (1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In P. J. Angeline and K. E. Kinnear, Jr. (eds.), *Advances in Genetic Programming 2*, 317-338. Cambridge, MA: MIT Press.

G. A Baker (1975). *Essentials of Padé Approximants*. New York: Academic Press.

C. M. Bender and S. A. Orszag (1978). *Advanced Mathematical Methods for Scientists and Engineers*. New York: McGraw-Hill.

T. Blickle and L. Thiele (1995). A comparison of selection schemes used in genetic algorithms. TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology.

R. L. Burden and J. D. Faires (1997). *Numerical Analysis*. Pacific Grove, CA: Brooks/Cole Publishing Company.

K. Chellapilla (1997). Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation* **1**(3):209-216.

L. Eulero (1734). De progressionibus harmonicus observationes. In *Comentarii academiæ scientarum imperialis Petropolitanæ* **7**(1734):150-161.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

G. H. Gonnet (1984). *Handbook of Algorithms and Data Structures*. London: Addison-Wesley.

M. A. Keane, J. R. Koza, and J. P. Rice (1993). Finding an impulse response function using genetic programming. In *Proceedings of the 1993 American Control Conference*, 3:2345-2350.

J. R. Koza (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University Computer Science Department technical report STAN-CS-90-1314.

J. R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

J. R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press.

S. Luke and L. Spector (1997). A comparison of crossover and mutation in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 240-248. San Mateo, CA: Morgan Kaufmann.

R. E. Moustafa, K. A. De Jong, and E. J. Wegman (1999). Using genetic algorithms for adaptive function approximation and mesh generation. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference*, 1:798. San Mateo, CA: Morgan Kaufmann.

P. Nordin (1997). *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik.

C. Ryan, J. J. Collins, and M. O'Neill (1998). Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (eds.), *Proceedings of the First European Workshop on Genetic Programming*, 1391:83-95. New York: Springer-Verlag.