18.337 Parallel Scientific Computing	February 14, 1995
Lecture 3	
Topic: The Pentium Bug	*1995 Scribe: Anthony Caola*

1 About These Notes on The Pentium Bug

The purpose of these notes is to summarize the 18.337 class on ♡ February 14, 1995 ♡. It includes a brief history on the discovery of the Pentium bug in §2 followed by the details of the Pentium bug flaw in §3. The material appearing in §3 is intended to be a simplification of the proof which appeared in the Coe/Tang paper "It Takes Six Ones to Reach a Flaw."

2 The Discovery of The Bug

The first posting about the Pentium bug was by Professor Nicely at the Lynchburg College of Virginia. It seems that Dr. Nicely was working with twin primes – numbers such as { 11, 13 }, { 29, 31 }, or { 107, 109 }. Specifically, he was going after the sum:

$$S = \frac{1}{5} + \frac{1}{7} + \frac{1}{11} + \frac{1}{13} + \frac{1}{17} + \frac{1}{19} + \cdots$$

This sum is mathematically known to be finite. Dr. Nicely had been running his program to compute this sum on several different computers. But when he ran the program on a Pentium machine, he got a different answer than he expected – even though he was using double precision, he was getting worse results than would be expected from single precision numbers. Eventually, Dr. Nicely narrowed the the source of the error to a part of his code that computed the terms in the sum. (i.e. $\frac{1}{p}$) He knew that his code was correct, so on October 10, 1994 he posted his conclusion that the Pentium chip contained a division flaw.

This posting caught the attention of Dr. Timothy Coe at Vitesse, who decided that this was a good opportunity to discover how the chip works. He asked for people to send him examples of the bug, and successfully put together a model of what intel was doing.

3 The Pentium Bug

What Dr. Coe discovered was that Intel was doing division using the Radix-4 version of the Sweeney, Robertson, and Tocher (SRT) algorithm. In order to get an idea of the magnatude of the bug that we will be discussing, consider the following example: The fraction $\frac{4195835}{3145727}$ should evaluate to 1.33382, but a flawed Pentium chip returns 1.33374. This is a relative error of 6.1E-5 or about 61 ppm – roughly twice the concentration of carbon monoxide allowed in ambient air by national ambient air quality standards [1]. It is interesting that some software

[1]National Primary and Secondary Ambient-Air-Quality Standard, 1983

developers have come up with ways to deal with this annoying bug. For example, there is a patch available for MATLAB which detects at-risk denominators, multiplies the numerator and denominator by $\frac{15}{16}$, and then performs the division. The remainder of this section is devoted to a description of the details of the Radix-4 SRT algorithm and the Pentium Bug.

3.1 Introduction to Radix-4 SRT

The SRT algorithm computes a radix 4 quotient where the digits are not the set $\{0, 1, 2, 3\}$ as might be expected from base 4, but rather $\{-2, -1, 0, 1, 2\}$.

Letting p and d denote the numerator and the denominator, we may as well assume that $1 \leq p, d < 2$. The SRT division algorithm may be expressed succintly in conceptual pseudocode:

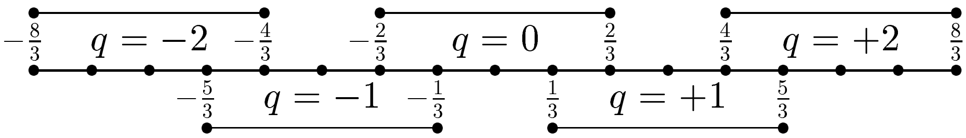
Radix 4 SRT Division

$p_0 := p$
for $k = 0, 1, \ldots$
  "Lookup" a digit $q_k \in \{-2, -1, 0, 1, 2\}$ in such a way that
    $p_{k+1} := 4(p_k - q_k d)$
  satisfies $|p_{k+1}| \leq \frac{8}{3}d$
end
$p/d = \sum_{i=0}^{\infty} q_i/4^i$.

On the Pentium chip, by an accident in the table lookup, it is possible to obtain a q_k which fails to produce an p_{k+1} with absolute value $\leq \frac{8}{3}d$.

3.2 Understanding why SRT works

It is easy to see that if $|p_k| \leq \frac{8}{3}d$ then at least one of the five values of $p_k - q_k d$ has absolute value $\leq \frac{2}{3}d$. The figure below (with unit length d) shows this clearly. The indicated q value translates each interval to the $q = 0$ interval. Overlap between upper and lower intervals indicates that an arbitrary choice may be made.



We therefore may run the algorithm ad infinitum, i.e. we can find a q_k for which

$$p_{k+1} = 4(p_k - q_k d) \tag{1}$$

has absolute value $\leq \frac{8}{3}d$. Define $Q_k \equiv \sum_{i=0}^{k-1} q_i/4^i$. The equation

$$\frac{p}{d} = Q_k + \left(\frac{p_k}{d}\right) 4^{-k} \tag{2}$$

may be proved easily by induction since Equation (1) is equivalent to

$$p_k 4^{-k} = p_{k+1}/4^{k+1} + q_k d/4^k.$$

Taking the limit as $k \to \infty$ of Equation (2) proves that we are computing the correct quotient. Here we used the fact that p_k/d is bounded by a constant (8/3 in our case).

A consequence of Equation (2) is that

$$p_k = d \sum_{i=k}^{\infty} q_i/4^{i-k} = d(q_k + \frac{q_{k+1}}{4} + \frac{q_{k+2}}{4^2} + \ldots).$$

Summing the geometric progression with the extreme choices of all $q_i = +2$ or all $q_i = -2$ shows that the requirement $|p_k| \leq \frac{8}{3}d$, is not an arbitrary choice, but a necessary ingredient in the algorithm. On the Pentium, for certain values of d and for certain values of p_k just a little smaller than $\frac{8}{3}d$, the chip supplies the value 0 for q_k rather than 2. The value of p_{k+1} would then be nearly $10d$ which is way outside of the range representable by our geometric progression. A consequence is that when the Pentium looks up 0 instead of 2, there is no way to recover.

A small observation that we will use later is

Lemma 3.1 *If $q_k = 2$, then $p_{k+1} \leq p_k$.*

Proof This is a simple consequence of the bound $p_k \leq \frac{8}{3}d$. □

3.3 Quotient digit selection on the Pentium

Because of the overlap illustrated in the figure above, a choice is made when there are two possible digit choices. This is a well-known advantage of the SRT algorithm.

The algorithm uses D (an integer multiple of 1/16) and P_k (an integer multiple of 1/8) to approximate d and p_k respectively. These numbers satisfy

$$P \leq p_k < P + \frac{1}{4} \text{ and } D \leq d < D_+ \equiv D + \frac{1}{16}. \tag{3}$$

Notice that D and D_+ are defined by the above inequalities, but two possible values of P are possible given p_k. The choice depends on how the carry and sum bits line up.

Given d if D has any of the five values $17/16, 20/16, 23/16, 26/26, 29/16$. then a flawed entry exists at the value $P_{\text{Bad}} = \frac{8}{3}D_+ - \frac{1}{8}$. According to Coe and Tang, when D is one of these five values, the digit q is selected when

$$P_q^{\text{Min}} \leq P \leq P_q^{\text{Max}} \tag{4}$$

according to the table below:

q	P_q^{Min}	P_q^{Max}
-2	$-\frac{1}{8} - \frac{8}{3}D_+$	$-\frac{1}{4} - \frac{4}{3}D_+$
-1	$-\frac{1}{8} - \frac{4}{3}D_+$	$-\frac{1}{4} - \frac{1}{3}D_+$
0	$-\frac{1}{8} - \frac{1}{3}D_+$	$-\frac{1}{8} + \frac{1}{3}D_+$
1	$\frac{1}{3}D_+$	$-\frac{1}{8} + \frac{4}{3}D_+$
2	$\frac{4}{3}D_+$	$-\frac{1}{8} + \frac{8}{3}D_+$

It is easy to check that the q chosen in this manner satisfies the constraints of the algorithm specified in Subsection 3.1.

For the moment I am skipping over the details of the carry-sum implementation presupposing familiarity with such details. For now, let me remind the reader that the implementation avoids carry propogation

3.4 Simplifying the proof

The Coe/Tang proof may be simplified and broken down into two parts. One is a simple analysis of the inequalities that either prevent you or allow you access to a buggy table entry, and the second is an arithmetic puzzle not so very different in spirit from the sort found in recreational mathematics, where one has to replace letters with digits to make a correct sum as in this one stolen from *Games for the Super-intelligent* by James Fixx:

```
  S E N D
+ M O R E
---------
M O N E Y
```

Equation (1) is replaced by

$$P_{k+1} = 4(P_k - q_k D_+) + R_k, \tag{5}$$

where R_k is determined by a few higher order bits. To be precise,

$$R_k = 0.0s_4s_5 + 0.0c_4c_5 + \left\{\begin{array}{l} 0.0\,d_5\,d_6 \\ 0.0\,0\;\;d_5 \\ 0 \\ 0.0\,0\;\;\bar{d}_5 \\ 0.0\,\bar{d}_5\,\bar{d}_6 \end{array}\right\} + \frac{1}{8} \times \text{carry-over}(s_6, c_6, \left\{\begin{array}{c} d_7 \\ d_6 \\ 0 \\ \bar{d}_6 \\ \bar{d}_7 \end{array}\right\}) + \left\{\begin{array}{c} -1/2 \\ -1/4 \\ 0 \\ 0 \\ 0 \end{array}\right\},$$

where the braces indicate the values for $-2, -1, 0, 1, 2$ respectively and the s_i, c_i, and d_i are the appropriate bits in the sum, carry, and divisor respectively. The function carry-over(b_1, b_2, b_3) is 1 if at least two of its arguments are 1. The last correction term of $-1/2$ and $-1/4$ for $q = -2, -1$ exists because we are expressing P_{k+1} in terms of D_+ rather than D. For $q = 1, 2$, the correction term is cancelled by the error that exists from using $\bar{D}$ rather than $\bar{D} - \frac{1}{16}$ for the value of $-D$.

Taking the maximum value of the parameters we see that

$$R_k \le R_k^{\text{Max}} \equiv \left\{\begin{array}{c} 3/4 \\ 3/4 \\ 3/4 \\ 1 \\ 5/4 \end{array}\right\}. \tag{6}$$

We assume that D_+ is a multiple of 3/16 i.e., it corresponds to one of the five flawed table entries. We may tabulate the largest possible P_{k+1} as a function of certain extreme P_k given q_k by using equations (5) and (6):

(7)

q_k	P_k	$P_{k+1}^{\text{Max}} = 4(P_k - q_k D_+) + R_k^{\text{Max}}$
-2	$P_{-2}^{\text{Max}} = -\frac{4}{3}D_+ - \frac{1}{4}$	$\frac{8}{3}D_+ - \frac{1}{4} = P_{\text{Bad}} - \frac{1}{8}$
-1	$P_{-1}^{\text{Max}} = -\frac{1}{3}D_+ - \frac{1}{4}$	$\frac{8}{3}D_+ - \frac{1}{4} = P_{\text{Bad}} - \frac{1}{8}$
0	$P_{0}^{\text{Max}} = \frac{1}{3}D_+ - \frac{1}{8}$	$\frac{4}{3}D_+ + \frac{1}{4} < P_{\text{Bad}} - \frac{1}{8}$
1	$P_{1}^{\text{Max}} = \frac{4}{3}D_+ - \frac{1}{8}$	$\frac{4}{3}D_+ + \frac{1}{2} < P_{\text{Bad}} - \frac{1}{8}$
2	$P_{\text{Bad}} - \frac{1}{8} = \frac{8}{3}D_+ - \frac{1}{4}$	$\frac{8}{3}D_+ + \frac{1}{4} > P_{\text{Bad}}$
2	$P_{\text{Bad}} - \frac{1}{4} = \frac{8}{3}D_+ - \frac{3}{8}$	$\frac{8}{3}D_+ - \frac{1}{4} = P_{\text{Bad}} - \frac{1}{8}$

The two "less than" inequalties in the table above are a simple consequence of $D_+ > 1$.

Lemma 3.2 *The sequence of P's and corresponding q's that lead to the bad entry are given below:*

$$\begin{array}{lccccc} P\text{:} & P_{-2}^{\text{Max}}/P_{-1}^{\text{Max}} & \rightarrow & \left\{P_{\text{Bad}} - \frac{1}{8}\right\}_{m \geq 1} & \rightarrow & P_{\text{Bad}} \\ q: & -2/-1 & \rightarrow & \{2\}_{m \geq 1} & \rightarrow & \text{(bad entry)} \end{array} . \tag{8}$$

Here the subscript m in the middle column denotes a repetition of the entry $m \geq 1$ times, and the first column indicates that either a P_{-2}^{Max} or P_{-1}^{Max} may start the path to the bug corresponding to either $q = -2$ or $q = -1$ respectively.

Proof Our proof goes from right to left. The table in (7) shows that we can only reach P_{Bad} from $P_{\text{Bad}} - \frac{1}{8}$. The table also shows that for $q = -2$ or $q = -1$, we can just barely reach $P_{\text{Bad}} - \frac{1}{8}$, but only when $P_k = P_k^{\text{Max}}$ and $R_k = R_k^{\text{Max}}$. $P_{\text{Bad}} - \frac{1}{8}$ may also be reached from $P_{\text{Bad}} - \frac{1}{4}$, but if $P_k = P_{\text{Bad}} - \frac{1}{4}$ then $P_{\text{Bad}} - \frac{1}{4} \leq p_k < P_{\text{Bad}}$, and Lemma 3.1 guarantees that while the sequence produces 2's all future p_k will also satisfy $p_k < P_{\text{Bad}}$ which precludes reaching the flawed entry, i.e., reaching k with $P_k = P_{\text{Bad}}$. $\square$

3.5 Possible bit patterns

We now solve the puzzle of which bit patterns may occur in the sum, carry, and divisor words as we head towards the bug. We first show that if $m = 1$, the divisor must satisfy $d_5 = d_6 = d_7 = d_8 = d_9 = 1$. We then show that $m > 1$ can not occur so the previous case was in fact the only case.

3.5.1 $m = 1$

The table below illustrates the carry-sum addition that leads to the buggy entry. Readers wishing to verify that these bit patterns are the only possible are invited to fill in the bits in the order of the Greek subscripts which are keyed to an explanation of why these bit patterns must appear as they do. We only fill in the details when the first q is -2, the -1 case is almost the same except for the position of the d_i bits.

$$
\begin{array}{ccccccccccccccccccc}
 & s_{j-2} & .x & x & x & 1_\alpha & 1_\alpha & 1_\zeta & 1 & 1 & s_{j-2} & .x & x & x & 1 & 1 & 1 & 1 & 1 & \\
\boxed{q=-2} & c_{j-2} & .x & x & x & 1_\alpha & 1_\alpha & 1_\zeta & 1 & 1 & c_{j-2} & .x & x & x & 1 & 1 & 1 & 1 & 1 & \boxed{q=-1} \\
 & 2d & .d_2 & d_3 & d_4 & 1_\alpha & 1_\alpha & 1_\zeta & 1 & 1 & d & .d_1 & d_2 & d_3 & d_4 & 1 & 1 & 1 & 1 & \\
\hline
 & s_{j-1} & .x & 1_\beta & 1_\beta & 1_\epsilon & 1_\iota & 1 & & & & & & & & & & & & \\
\boxed{q=2} & c_{j-1} & .1_\beta & 1_\beta & 1_\gamma & 1_\epsilon & 1_\iota & 1 & & & & & & & & & & & & \\
 & -2d & .\bar{d}_2 & \bar{d}_3 & \bar{d}_4 & 0_\delta & 0_\delta & 0_\kappa & & & & & & & & & & & & \\
\hline
 & s_j & .x & 0_\eta & 0_\lambda & 0 & & & & & & & & & & & & & & \\
\boxed{\text{bug}} & c_j & .1_\eta & 1_\theta & 1_\lambda & x & & & & & & & & & & & & & &
\end{array}
$$

- α Since R must equal $R^{\max}$ we must have $c_4, c_5, s_4, s_5, d_5, d_6 = 1$.
- β Arithmetic from above.
- γ Two explanations: 1) $8(P_{\text{Bad}} - \frac{1}{8})$ is even or 2) we must have a carry from above for the same reason as in α.
- δ Complement d bits found in α.
- ϵ R must equal $R^{\max} - \frac{3}{8}$. Given δ, if either 1_ϵ were 0, R would be smaller.
- ζ Arithmetic from below.
- η Arithmetic from above.
- θ $8P_{\text{Bad}}$ is 3 (mod 4).
- ι The 1_θ requires a carry from above.
- κ Complement the d bit 1_ζ.
- λ 0_λ is the result of arithmetic from above. 1_λ since $8P_{\text{Bad}}$ is 3 (mod 4).

The remaining ones follow easily.

The conclusion that we have so far is that if we begin with $q = -2$, we obtain d_5, d_6, d_7, d_8, d_9 are all 1. If we begin with $q = -1$, we only obtain d_5, d_6, d_7, d_8 are all 1.

Having all those ones at the $q = -2$ or $q = -1$ step allows us to easly go back yet another step and obtain two more ones giving the at least six claimed by Coe and Tang.

3.5.2 $m > 1$

Details omitted for now, but the same sort of bit chasing yields a contradiction. Stay tuned.

3.6 At least nine steps to failure

We now show that on the Pentium that $q_1, \ldots, q_8$ are always computed correctly. Examples are known such as $n =$`800bf7` and $d =$`bffffe` (in hex), where the flawed table entry gives an incorrect q_9. While it is difficult to make claims as to who may have been hurt by the Pentium bug, it is safe to say that the smaller the largest possible computed relative error, the less likely the bug causes harm.

The proof that the bad digit can not occur at or before q_8 is a straightforward consequence of the results of Section 6.1 In particular. If the bug occurs at step 8, then we saw in Section 6.1 that step 6 must have the form shown in the diagram below which shows the sum, carry, and $q_k d$ bits respectively starting with bit number 4. Taking into account that all the carry bits are 0 at step 1, it is fairly easy to show that the bit patterns must fall as in the diagram below. However, since q_1 is positive, it must follows that all of $q_2, \ldots, q_6$ are positive by observing the overlap in the $q_k d$ row from one step to the next. However, we known that $q_6 < 0$ so we have a contradiction. Therefore the pattern shown below as Step 6, which must occur to trigger the bug, can not appear before the seventh step, and hence the bug appears no earlier than the ninth step.

	· · · · · · · · · · · · · · · 1 1 1 1 1
Step 1:	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	· · · · · · · · · · · · · · · 1 1 1 1 1
	· · · · · · · · · · · · 1 0 0 0 0 0 · ·
Step 2:	· · · · · · · · · · · · 1 1 1 1 1 · · ·
	· · · · · · · · · · · · 1 1 1 1 1 · · ·
	· · · · · · · · · 1 1 0 0 0 0 · · · · ·
Step 3:	· · · · · · · · · 1 1 1 1 1 · · · · · ·
	· · · · · · · · · 1 1 1 1 1 · · · · · ·
	· · · · · · 1 1 1 0 0 0 · · · · · · · ·
Step 4:	· · · · · · 1 1 1 1 1 · · · · · · · · ·
	· · · · · · 1 1 1 1 1 · · · · · · · · ·
	· · · 1 1 1 1 0 0 · · · · · · · · · · ·
Step 5:	· · · 1 1 1 1 1 · · · · · · · · · · · ·
	· · · 1 1 1 1 1 · · · · · · · · · · · ·
	1 1 1 1 1 · · · · · · · · · · · · · · ·
Step 6:	1 1 1 1 1 · · · · · · · · · · · · · · ·
	· 1 1 1 1 · · · · · · · · · · · · · · ·

↑
Bit #4

3.7 Viewing SRT as a dynamical system

Let I_D denote the interval $[-\frac{8}{3}D, \frac{8}{3}D]$. We may create an SRT algorithm by defining a map f_D from I_D into I_D such that $f_D(x) = 4(x - q(x)D)$, where $q(x)$ is in our digit set. The word "into" means that we require that $f_D(x)$ be in the interval I_D, i.e. the graph of f must fall entirely inside the box $I_D \times I_D$. Figures 1 and 2 below show the choices that the

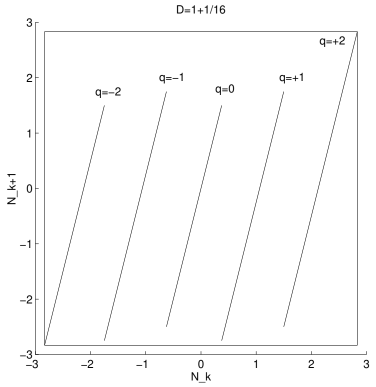


Figure 1: $f_D(x)$ where $D = 1 + \frac{1}{16}$

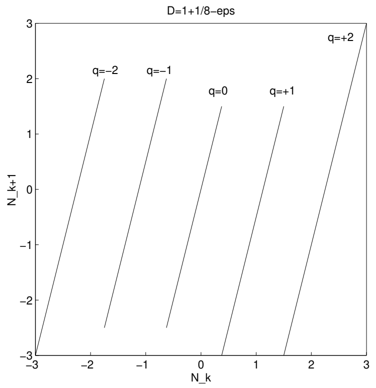


Figure 2: $f_D(x)$ where $D = 1 + \frac{1}{8} - \epsilon$

Intel Pentium makes for two different values of D. The function $f_D(x) = 4(x - q(x)D)$ is piecewise linear with five segments all of whose slopes are exactly equal to 4. The function is confined to the box $I_D \times I_D$ which is slightly smaller in Figure 1 than Figure 2 because the value of D is smaller in Figure 1.

Given an initial $x \in [1, 2)$, the division algorithm we boxed in Section 4 may be realized as a repeated iteration of the map f_D, where at each step the q_i is recorded. Specialists in dynamical systems study repeated iterations of maps, but I suspect there is very little literature on the maps illustrated in the above two figures.

3.8 Some Pentium Implementation Details

4 Some Pentium implementation details

Of course the Pentium does not store a graph, but rather a lookup table. Most likely this lookup table exists not as a table at all, but rather as logic on the chip. How it exists is not important, that it has five bad entries is.

Though the graphs in Figures 1 and 2 depend explicitly on D, it is possible to construct a table that picks a valid q_i based on crude representations of the N_k and the D. The table on the last page of this document shows the format of the table and valid entries that would keep the graph inside the box $I_D \times I_D$. This table does not list the explicit entries in the table but rather which entries would be valid. I believe I know the exact entries, but this is not relevant here. Note also that there is a negative half to the table that is not shown.

Notice that divisors are broken up by increments of 1/16 and the shifted remainder (i.e., the Q_k) by increments of 1/8. The bad entries are indicated by the five shaded boxes. Figures 1 and 2 both take place in the region indicated by the second column $17/16 \leq D \leq 9/8$. Though the graphs in Figures 1 and 2 are different, the breakpoints between the lines are the same. This is what the table really looks up when performing the computation.

Notice that in Figures 1 and 2, some of the five line segments do not touch the boundary of the box $I_D \times I_D$. This means that these lines may be extended and we may still have a valid algorithm.

What happens in practice is that the Pentium does not use the exact value of N_k to choose q_k. Rather it computes a value for N_k that is guaranteed to be within the interval $[N_k - 1/4, N_k]$. I can not overemphasize that this a clever idea, not a bug! The algorithm picks q based on this approximate value, but computes N_{k+1} to high accuracy. What actually occurs is that N_k is stored in what is known as "Carry-Save" format. The details of this format are not important here, but it is worth knowing that this gives a fast method for computing N_{k+1} from N_k, q_k, and D. The first few bits of both the carry and save numbers are used to approximate N_k. Ultimately the N_k that is used to pick the q_k is a number that is an integer multiple of 1/8. In the most extreme case 1/8 may be lost from the carry, and 1/8 from the sum giving an approximation error of 1/4.

The effect of this approximation is that sometimes the algorithm chooses the solid lines in Figures 3 and 4, as opposed to the lines from Figures 1 and 2, which are now shown dotted.

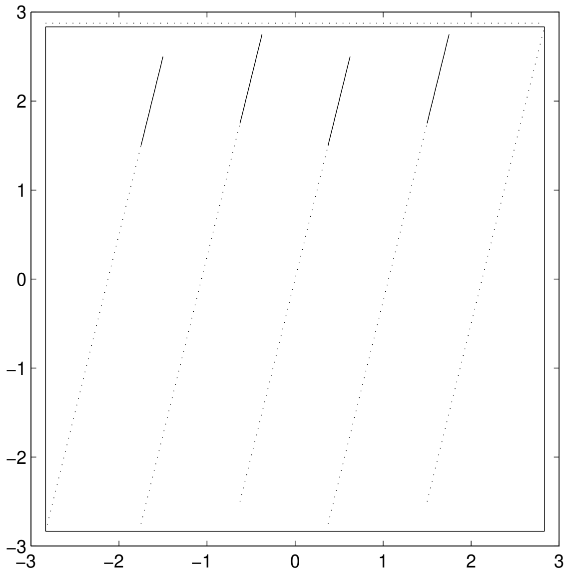


Figure 3: $f_D(x)$

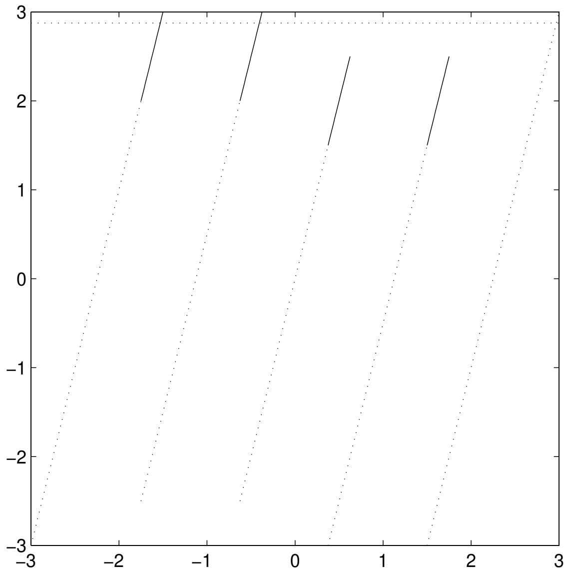


Figure 4: $f_D(x)$

5 Postscript

I have more material on this subject.

References

[1] Tim Coe and Ping Tak Peter Tang. *It Takes Six Ones To Reach a Flaw.* 1995.

[2] Alan Edelman. *Pentium Bug – a glimpse at the technical details.* 1995.

[3] Alan Edelman. *Remarks on the Coe/Tang Paper: It Takes Six Ones to Reach a Flaw.* 1995.

[4] David Goldberg. *Tutorial on Computer Division.* 1995.

[5] MIT Information Systems. *Essential LaTeX on Athena, Version 6.* IS Training and Publications Services, Cambridge, MA, 1994.