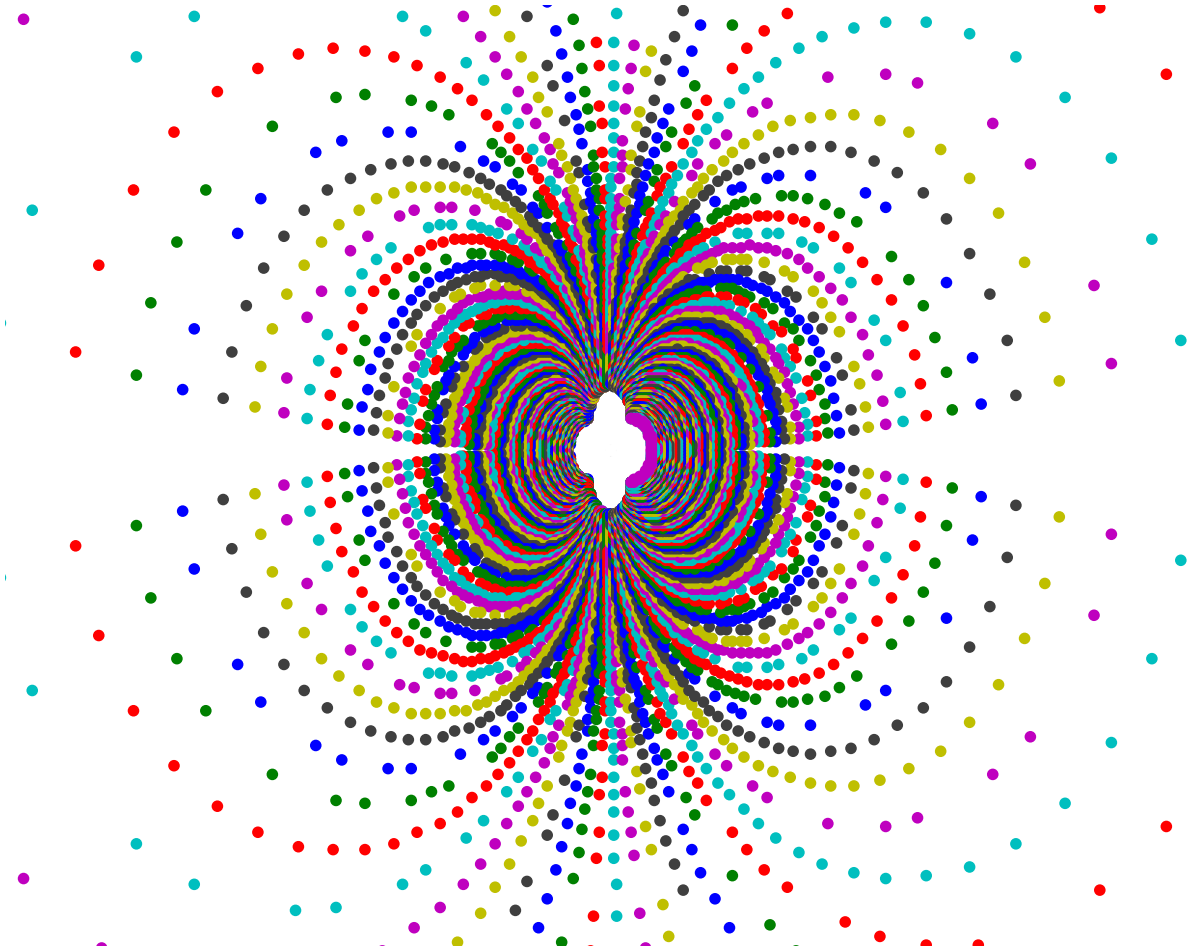
# Computation Of The Arithmetic-Geometric Mean

Tomack Gilmore
080106489



Queen Mary, University of London

# Contents

# 1 Introduction

The arithmetic-geometric mean of two numbers $a$ and $b$ is the common limit of the sequences $\{a_n\}$ and $\{b_n\}$ as $n$ tends to infinity, where each $a_n$ term is the arithmetic mean of $a_{n-1}$ and $b_{n-1}$ and each $b_n$ term is the geometric mean of $a_{n-1}$ and $b_{n-1}$ for $n \geqslant 1$. This report aims to explore and analyse the behaviour of this mean and some of its applications from a computational perspective.

We begin by considering the arithmetic-geometric mean of real numbers which is a single-valued function. We consider first the rate at which this function converges. From there we explore how the arithmetic-geometric mean can be used to calculate $\pi$ and we will see that this method is in fact the most efficient way of calculating this constant to high levels of accuracy.

We then consider the arithmetic-geometric mean on the complex plane, which is multi-valued. Once we relax the condition that $a$ and $b$ are real our algorithm generates uncountably many sequences as at each $n$th iteration we have a range of values from which we can choose $b_n$. In fact it turns out that countably many of these sequences have a non-zero limit and there is a formula that will generate every possible such limit for given $a$ and $b$. We generate these values and plot them in order to analyse their behaviour. We then plot the trajectories of these sequences after $n$ iterations to try and understand how our choice of values for $b_n$ at each iteration may effect the common limit of each pair of sequences. This has some surprising implications.

Many of the formulae in this report are taken from results that can be found in two papers- "The Arithmetic-Geometric Mean of Gauss" by David Cox (*L'Enseignement Mathematique*, Vol. 30, 1984, pp. 275-330), and "Gauss, Landen, Ramanujan, the Arithmetic-Geometric Mean, Ellipses, $\pi$, and the *Ladies Diary*" by G. Almkvist and B. Berndt (*The American Mathematical Monthly*, Vol. 95, 1988, pp. 585-608). Although inspired by these papers, the writing and code in this report are original, so they will not be cited further.

## 2 Real Numbers

The arithmetic-geometric mean for given $a, b \in \mathbb{R}_+$ where $a \geqslant b$ is the common limit of the sequences $\{a_n\}$ and $\{b_n\}$ as $n$ tends to infinity. Each term of the sequence is defined recursively in the following way:

$$\begin{aligned} a_0 &= a, & b_0 &= b, \\ a_n &= \frac{a_{n-1} + b_{n-1}}{2}, & b_n &= \sqrt{a_{n-1} b_{n-1}} \quad \text{for } n = 1, 2, ... \end{aligned}$$

The arithmetic-geometric mean inequality,

$$\frac{a+b}{2} \geqslant \sqrt{ab},$$

for $a, b \in \mathbb{R}_+$ where $a \geqslant b$ tells us immediately that $a_n \geqslant b_n$ for every $n$. Moreover it is also possible to show that $a_n \geqslant a_{n+1}$ and $b_n \leqslant b_{n+1}$ for each $n$ and that these sequences converge to a common limit. This common limit is the arithmetic-geometric mean of $a$ and $b$ and is denoted $M(a, b)$.

It is straightforward to create a function (agm.m) that will calculate this mean after $n$ iterations:

```
function[u,v]=agm(a,b)
%This function simply outputs a 1x2 matrix with the first entry being the
%arithmetic mean of a and b and the second entry the geometric mean

if a<=0|b<=0;
    error('agm:argchk','Both arguments must be positive real numbers')
else if a<b
    error('agm:argchk', 'First argument must be >= second argument')
end
u=(a+b)/2; %Calculates arithmetic mean
v=sqrt(a*b); %Calculates geometric mean
end
```

We call this function inside a procedure (realproc.m) that plots the trajectory of the $\{a_n\}$ and $\{b_n\}$ sequences for given $a$ and $b$ by generating an $n$x2 matrix $x$ in which the first column is the first $n$ terms of the $\{a_n\}$ sequence and the second column is the firs $n$ terms of the $\{b_n\}$ sequence:

```
a=200; %Initialises value of a.
b=5; %Initialises value of b.
n=7; %Sets number of iterations.
nspan=[0:n]; %Generates vector of iterative steps.
```

```
x(1,1)=a; %Sets (1,1)-entry of vector x as a.
x(1,2)=b; %Sets (1,2)-entry of vector x as b.

for i=1:n
    [s,t]=agm(x(i,1),x(i,2)); %Generates arithmetic and geometric mean of
                              %entries in previous row.
    x(i+1,1)=s; %Sets (i+1,1)-entry as arithmetic mean of x(i,1) and x(i,2).
    x(i+1,2)=t; %Sets (i+1,2)-entry as geometric mean of x(i,1) and x(i,2).
end

plot(nspan,x) %Generates plot of trajectories.
```

This produces the plots found in Figure 1, which show the trajectories of the sequences for various $a$ and $b$. It is clear from these plots that $b_n \leqslant b_{n+1}$ and $a_n \geqslant a_{n+1}$ for every $n$. Also it is immediately obvious that these sequences converge to a common limit and moreover they appear to converge at a very fast rate. In fact the difference between our initial $a$ and $b$ does not seem to affect the rate of convergence very much at all. It looks as though the sequences converge to a good degree of accuracy within 2 or 3 iterations. To see this mathematically consider the sequence $\{c_n\}_{n=0}^{\infty}$ where each $c_n$ is given by $c_n = \sqrt{a_n^2 - b_n^2}$. Then we have

$$c_{n+1} = \sqrt{\left(\frac{a_n + b_n}{2}\right)^2 - a_n b_n} = \frac{a_n - b_n}{2} = \frac{a_n^2 - b_n^2}{4a_{n+1}} \leqslant \frac{c_n^2}{4M(a,b)},$$

so it follows that the sequences converge quadratically, or alternatively convergence is of the second order.

It is possible to exploit the relationship between the arithmetic-geometric mean and elliptic integrals of the first kind via Legendre's relation to derive the following formula:

$$\pi = \frac{(2M(1,1/\sqrt{2}))^2}{1 - \sum_{n=1}^{\infty} 2^{n+1} c_n^2},$$

where $c_n$ is defined as above. Due to the quadratic rate of convergence of the arithmetic-geometric mean this provides us with an incredibly efficient method for calculating $\pi$ to a high degree of accuracy. We do this by calling piproc.m which can be obtained by modifying the previous procedure as follows. Our matrix $x$ is now a $n$x3 matrix where the third column is the $c_n$th term as defined above. We also introduce a value eps which is the maximum difference between the final terms in our sequences $\{a_n\}$ and $\{b_n\}$. So instead of looping for a specified number of iterations we now loop until the difference between the $a_n$th and $b_n$th term is less than the value eps:

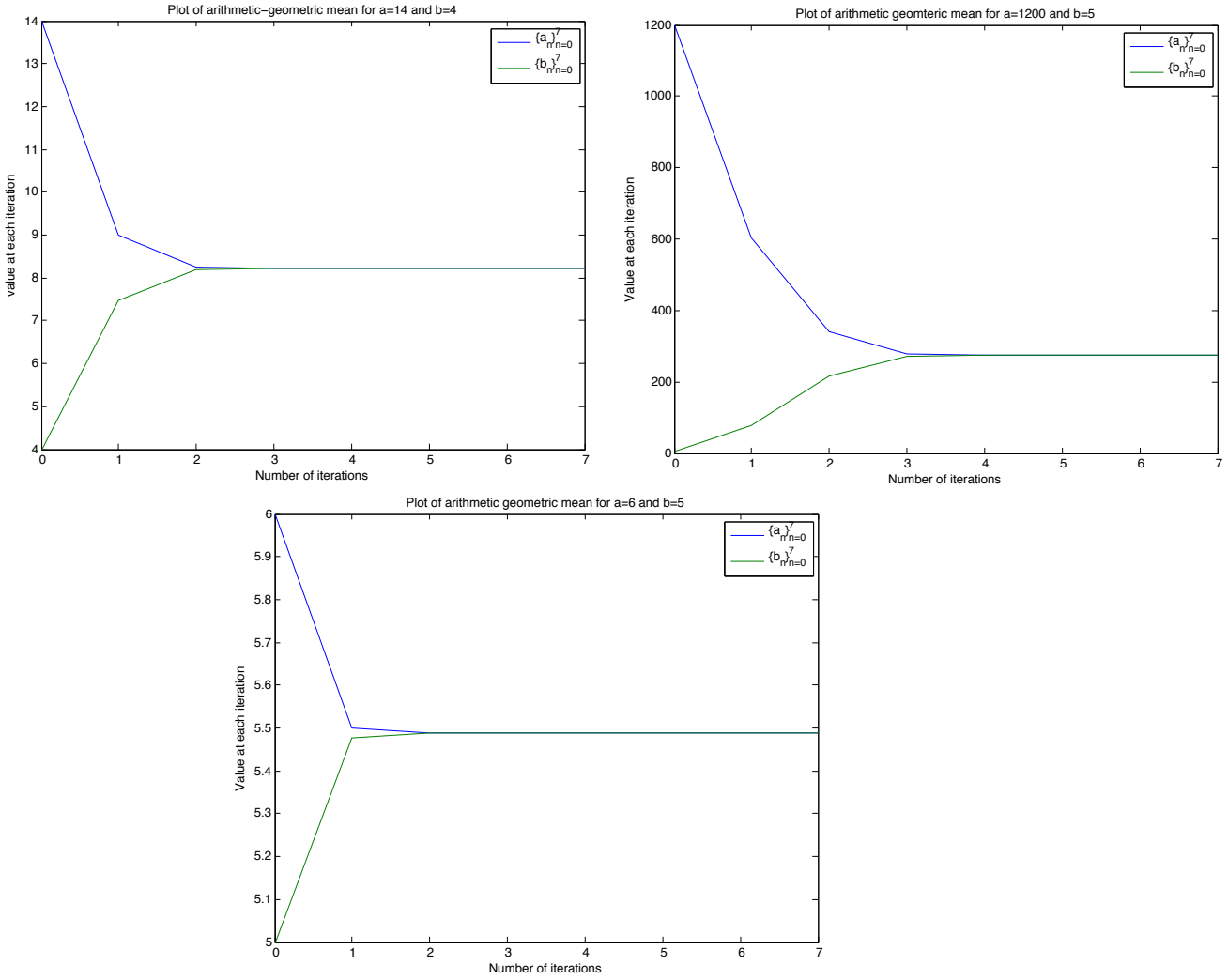

Plot of arithmetic–geometric mean for a=14 and b=4
value at each iteration
Number of iterations
Plot of arithmetic geomteric mean for a=1200 and b=5
Value at each iteration
Number of iterations
Plot of arithmetic geometric mean for a=6 and b=5
Value at each iteration
Number of iterations

Figure 1

```
a=1; %Sets a as 1.
b=1/sqrt(2); %Sets b as 1.
i=1; %Sets i as 1.
eps=10^(-15); %Sets the size of the error.
s=1; %Initialises s.
t=2; %Initialises t.

x(1,1)=a; %Sets (1,1)-entry of x as a.
x(1,2)=b; %Sets (1,2)-entry of x as b.
x(1,3)=(a^2-b^2); %Sets (1,3)-entry of x as a^2-b^2=c_0^2.

while abs(s-t)>=eps %Loops while the absolute difference between means is
                    %greater than eps.
    [s,t]=agm(x(i,1),x(i,2)); %Generates arithmetic mean (s) and geometric.
                              %mean (t) of x(i,1) and x(i,2).
    x(i+1,1)=s; %Sets (i+1,1)-entry of x as s.
    x(i+1,2)=t; %Sets (i+1,2)-entry of x as t.
    x(i+1,3)=(x(i+1,1)^2-x(i+1,2)^2); %Calculates c_(i+1).
    i=i+1; %Increments i by 1, counting the number of iterative steps.
end
n=i; %Sets n as number of iterations

su=0; %Initialises sum as zero.

for i=2:n %Runs over rows of x.
   su=su+(2^(i))*x(i,3); %Sums 2^(i+1)*x(i+1,3) over i.
end

p=4*x(n,1)^2/(1-su) %Approximates pi using formula.
```

By iterating only 4 times this produces an estimate of $\pi$ correct to 13 decimal places. We can see this by executing the following command

```
>> abs(pi-p)

ans =

      3.81916720471054e-14
```

We can see just how effective this method is by producing a plot of the logarithm of the absolute error between $\pi$ and the output of the procedure piproc.m ($p$) for successive iterations. It is also interesting to see whether our choice of $a_n$ or $b_n$ after $n$ iterations makes much of a difference. The plot in Figure 2 was generated by the following procedure

(log1.m):

```
for i=2:n %Runs loop from 2 to n
    su=0; %Initialises sum as zero
    for j=2:i %Runs over entries in third column of matrix x
        su=su+(2^j)*x(j,3); %Calculates the sum from the denominator of the
                            %equation for pi as the number of iterations increases
    end
    pa=4*x(j,1)^2/(1-su); %Estimates pi using the (i+1)th entry in first
                          %column of x as M(1,1/sqrt(2))
    y(i-1,1)=abs(pi-pa); %Calculates absolute error of pa
    pn=4*x(j,2)^2/(1-su); %Estimates pi using the (i+1)th entry in second
                          %column of x as M(1,1/sqrt(2))
    y(i-1,2)=abs(pi-pn); %Calculates absolute error of pn
end

plot([1:4],log(y(:,1)),'+',[1:4],log(y(:,2)),'x') %Generates plot
```

Figure 2 shows that even after the first iteration we have a very small error, and as the number of iterations increases the size of the error decreases dramatically as expected. We also see that there is almost no difference between whether we use the $n$th term of the $b_n$ sequence or the $a_n$ sequence.

Given that the sequences $\{a_n\}$ and $\{b_n\}$ are quadratically convergent it follows that at the $n$th iteration $a_n$ and $b_n$ agree for roughly the first $2^n$ decimal places. So in theory after $n$ iterations, our estimate of $\pi$ will be exact up to roughly the first $10^{-2n}$ decimal places. However we are limited by the fact that MATLAB works with up to 16 decimal points, our approximation of $\pi$ will only ever be correct to 13 decimal places. Indeed if we change eps to be $10^{-16}$ then the procedure will run endlessly without producing a result. In fact it is possible to change the procedure so that instead of a 'while' loop we run a 'for' loop over $n$ iterations. It appears that whatever $n > 4$ we choose we are limited by the level of precision that MATLAB uses. The following is the resulting matrix $x$ from piproc.m after six iterations:

```
>> x

x =

                   1     0.707106781186547                   0.5
   0.853553390593274     0.840896415253715    0.0214466094067263
   0.847224902923494     0.847201266746891    4.00497561866553e-05
   0.847213084835193     0.847213084752765    1.39667388765474e-10
   0.847213084793979     0.847213084793979    2.22044604925031e-16
   0.847213084793979     0.847213084793979    2.22044604925031e-16
```

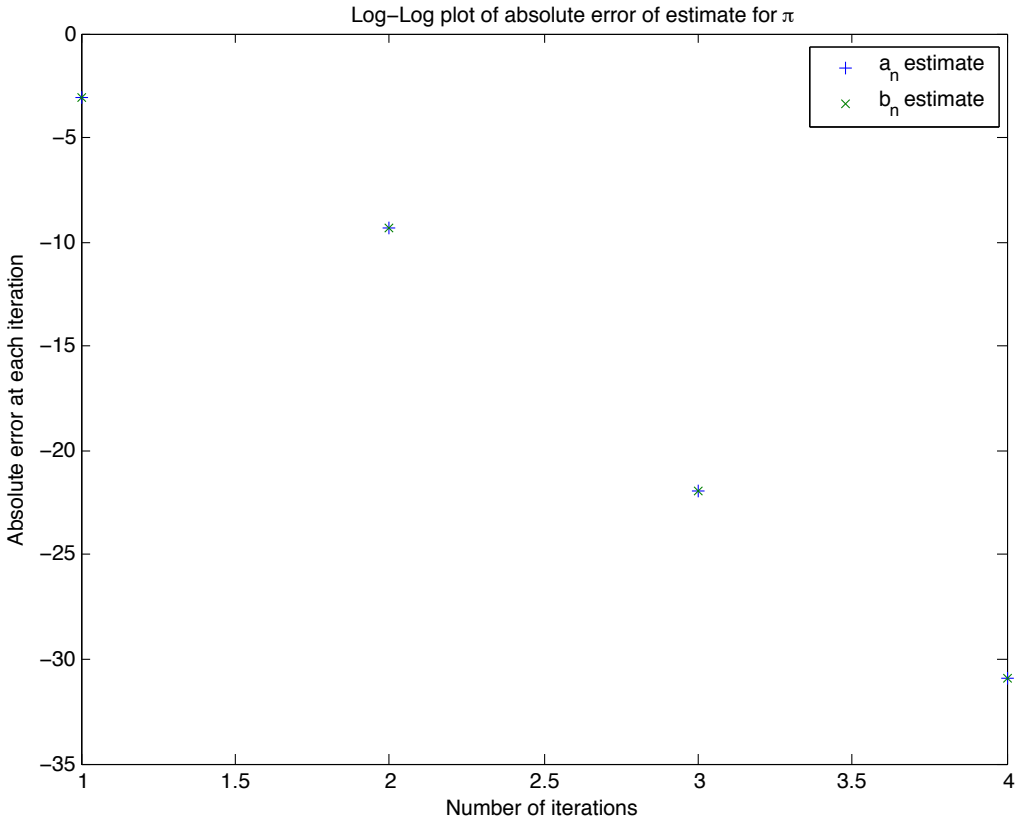

Log−Log plot of absolute error of estimate for π
Absolute error at each iteration
Number of iterations
a_n estimate
b_n estimate
0
−5
−10
−15
−20
−25
−30
−35
1
1.5
2
2.5
3
3.5
4

Figure 2

0.847213084793979 0.847213084793979 2.22044604925031e−16

So with this procedure the best approximation we can achieve for $\pi$ is correct to 13 decimal places. However this relationship between the arithmetic-geometric mean and $\pi$ provides us with the most efficient method for calculating this elusive number. To see this we may consider a different formula for $\pi$, attaributed to Francois Viete, that can be derived using Euler's formula:

$$\frac{\sin\theta}{\theta} = \prod_{n=1}^{\infty} \frac{\cos\theta}{2^n},$$

where the left hand side is interpreted to be 1 when $\theta = 0$. If we let $\theta = \pi/2$ then the above formula becomes

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots .$$

This may then be written as

$$\frac{2}{\pi} = \lim_{n\to\infty} 2^{-n} \prod_{i=1}^{\infty} a_i,$$

where $a_0 = 0$ and $a_i = \sqrt{2 + a_{i-1}}$ for $i \geqslant 1$. This is a very straightforward procedure to compute $\pi$, however it converges at a much slower rate than our previous algorithm. To see this we call the function viete.m, which calculates $\pi$ for $a_1, ..., a_n$. We know that our procedure piproc.m produces an approximation of $\pi$ correct to 13 decimal places after just 4 iterations. So in order to compare the algorithms we let $n = 4$ in the following procedure:

```
a=0; %Initialises a.
n=4; %Sets number of iterations.
p=1; %Initialises p.

for i=1:n %Runs from 1 to n.
    a=sqrt(2+a); %Calculates a_i.
    p=p*a/2; %Calculates p_i.
    err(i)=abs(pi-1/(p/2)); %Calculates absolute error between pi and
                            %the estimate produced by the formula.
end

nspan=1:n %Generates vector of values from 1 to n.

plot(nspan,log(err),'x',[1:4],log(y(:,2)),'+') %Generates plot.
```

This produces the log plot found in Figure 3. We can clearly see from the plot that the absolute error decreases far quicker for the arithmetic-geometric mean approximation than for the Viete approximation. In fact it is possible to alter the procedure viete.m so

that instead of a 'for' loop running from 1 to $n$ we run a 'while' loop and specify the size of the error we require. We do this by calling the procedure vietealt.m:

```
a=0; %Initialises a.
p=1; %Initialises p.
eps=10^(-15); %Sets size of absolute error.
i=1; %Initialises i.
est=0; %Initialises est.

while abs(pi-est)>=eps %This loops until the difference between pi
                       %and our estimate (est) is less than eps.
    a=sqrt(2+a); %This calculates each a_i.
    p=p*a/2; %This calculates the product in the formula.
    est=1/(p/2); %This calculates the estimate using the product p.
    i=i+1; %This counter keeps track of the number of iterations.
end
```

Once this procedure has been called we see that $i = 26$, so in order to get roughly the same level accuracy as that of piproc.m we must iterate 26 times. This is insignificant when working with a system that works only to 16 decimal points, so either method is just as efficient in MATLAB. If we were to try to calculate $\pi$ accurate to say 1000 decimal points, however, it should be clear that the method using the arithmetic-geometric mean will converge much quicker.

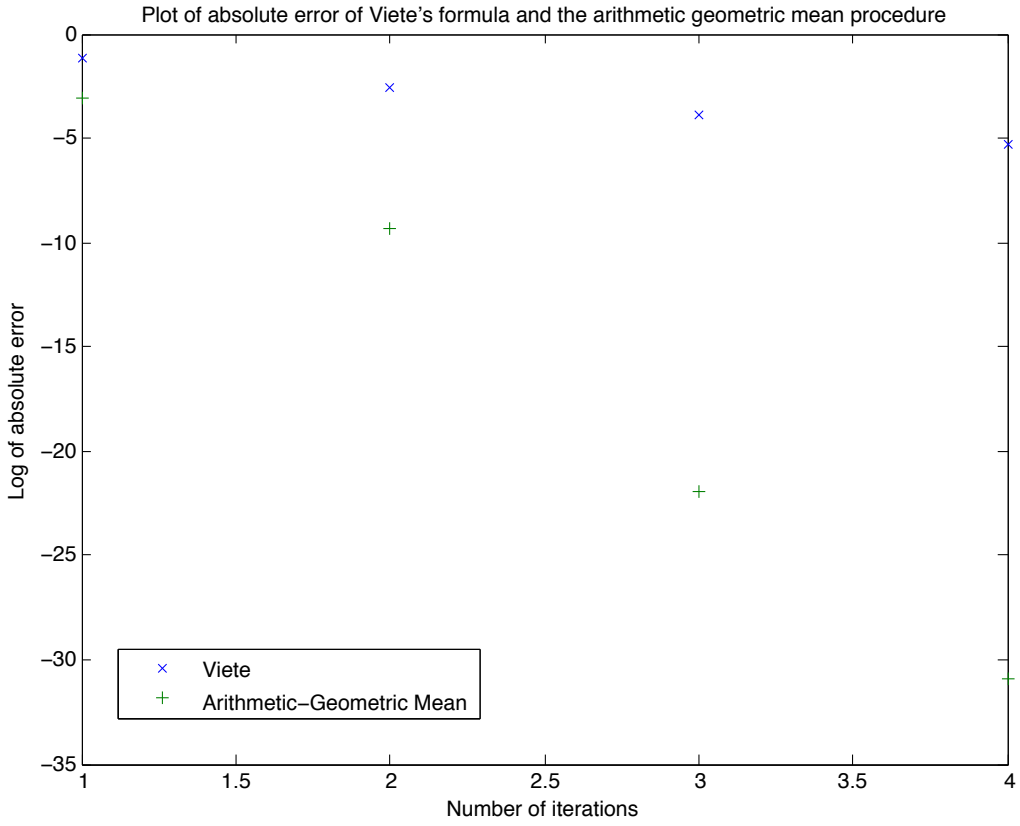

Plot of absolute error of Viete's formula and the arithmetic geometric mean procedure
Log of absolute error
Number of iterations
0
−5
−10
−15
−20
−25
−30
−35
1
1.5
2
2.5
3
3.5
4
Viete
Arithmetic−Geometric Mean

Figure 3

## 3 The Complex Plane

If we now instead consider $a, b \in \mathbb{C}$ it is no longer clear whether the sequences $\{a_n\}$ and $\{b_n\}$ converge to a common limit. This is because we now have a number of possible choices for the value of each $b_n$. So after $n$ iterations we have $2^n$ possible values of $b_n$ and consequently $2^{n-1}$ possible values for $a_n$. In order to combat this we impose conditions that enable us to categorise each value of $b_n$. Given $a, b \in \mathbb{C}$, we say that the geometric mean of $a$ and $b$, $b_1$, is the *right choice* if the following conditions hold:

1. $|a_1 - b_1| \leqslant |a_1 + b_1|$;
2. whenever $|a_1 + b_1| = |a_1 - b_1|$ we have $\Im(b_1/a_1) > 0$,

where $\Im(a)$ denotes the imaginary part of $a$. Once this has been defined it is possible to show that given $a, b \in \mathbb{C}$ every pair of sequences generated by always making the right choice for $b_n$ for $n > 0$ will converge to a common limit. However it is possible that we may have started at some other $a'$ and $b'$ in the complex plane and arrived at $a$ and $b$ via our algorithm having not necessarily always made the right choice for $b_n$, but if we then continued to make the right choice for every $b_n$ the sequences would still converge. We call sequences such as this *good sequences*, and they are formally defined as sequences $\{a_n\}$ and $\{b_n\}$ where $b_n$ is the right choice for all but finitely many $n \geqslant 0$. It has been shown that every good sequence converges to a non-zero limit, whilst every sequence that is not good converges to zero. We distinguish the limit of a sequence in which every value of $b_n$ is the right choice as the *simplest value*.

There is a theorem that enables us to generate every possible arithmetic-geometric mean, $\mu'$, of some given $a, b \in \mathbb{C} \setminus \{0, 1\}$ with $|a| > |b|$. The formula is

$$\frac{1}{\mu'} = \frac{d}{\mu} + \frac{ic}{\lambda}, \tag{1}$$

where $\mu$ and $\lambda$ are the simplest values of $(a, b)$ and $(a + b, a - b)$ respectively, $c \equiv 0 \bmod 4$, $d \equiv 1 \bmod 4$ and $\gcd(c, d)$=1.

In order to compute these values we define a function that will calculate the simplest value of two complex numbers. The following function 'agmsimp.m' calculates such a value for a given number of iterations, $n$, and outputs two vectors, $an$ and $bn$, whose final entries agree to a high degree of accuracy:

```
function [an,bn]=agmsimp(a,b,n)
%This function finds the simplest value of the arithmetic-geometric mean
%for two real or complex numbers a and b and outputs a vector of the values
%in the sequence.
an(1)=a; %Sets first entry of vector an as a.
bn(1)=b; %Sets first entry of vector bn as b.
```

```
for j=1:n %Runs from 1 to n iterations.
     [s,t]=agmc(an(j),bn(j)); %Calculates agm of jth entris of an and bn.
     an(j+1)=s; %Sets entry an(j+1) as arithmetic mean of an(j) and bn(j).
     bn(j+1)=t; %Sets entry bn(j+1) as geometric mean of an(j) and bn(j).
          if (abs(s-t)<=abs(s+t))||(abs(s-t)==abs(s+t)&&imag(t/s)>0) %This
              %condition checks if the value t is the right choice for b_n.
              bn(j+1)=t; %If t is the right choice, we assign t to bn(j+1).
          else
              bn(j+1)=-t; %If t is not the right choice then -t is, and is
              %assigned to bn(j+1).
          end
end
end
```

The function 'agmc.m' that we call in line 9 is the same as the function 'agm.m' from section 2 without the conditional error messages included:

```
function[u,v]=agmc(a,b)
%This function simply outputs a 1x2 matrix with the first entry being the
%arithmetic mean of a and b and the second entry the geometric mean.
u=(a+b)/2; %Calculates arithmetic mean.
v=sqrt(a*b); %Calculates geometric mean.
end
```

With these functions we can proceed to generate some of the arithmetic-geometric means of given $a$ and $b$. We do this by specifying a range for $c$ and $d$ and then generating vectors whose entries are elements of these ranges. We then calculate the simplest values for $\mu$ and $\lambda$, 'mu' and 'lam', and generate a matrix $x$ in which the columns are all arithmetic-geometric means for fixed $d$ and varying $c$. Likewise the rows of $x$ are all arithmetic-geometric means for fixed $c$ and varying $d$, where $c$ and $d$ are relatively prime. We then generate a plot of these values. We do this by calling the procedure 'means.m':

```
a=12+32*1i; %Sets value of a.
b=2-1i; %Sets value of b.
cmin=-400; %Sets minimum of range of c.
cmax=400; %Sets maximum of range of c.
c=cmin:4:cmax; %Sets c as a vector of entries =0 (mod 4).
dmin=-403; %Sets minimum of range of d.
dmax=397; %Sets maximum of range of d.
d=dmin:4:dmax; %Sets d as a vector of entries =1 (mod 4)

[mu1,mu2]=agmsimp(a,b,10); %Generates simplest value of a, b after 10
                           %iterations.
```

```
mu=mu1(11); %Sets final entry of mu1 as value for mu.

[lam1,lam2]=agmsimp(a+b,a-b,10); %Generates simplest value of a+b, a- b
                                 %after 10 iterations.
lam=lam1(11); %Sets final entry of lam1 as value for lambda.

x=zeros(length(c),length(d)); %Generates zero matrix for efficiency.
t13=zeros(length(c),length(d)); %Generates zero matrix for efficiency.

for j=1:length(c) %Runs over entries of c.
    for k=1:length(d) %Runs over entries of d.
        if gcd(c(j),d(k))==1 %Checks if jth entry of c is co-prime to kth
                             %entry of d.
            x(j,k)=1/(d(k)/mu+1i*c(j)/lam); %Calculates mu' for each c,d.
        end
        t13(j,k)=n+n1; %This is useful for plotting in 3d.
    end
end


plot(real(x),imag(x),'.') %Generates 2d plot of each mu' on complex plane.
```

Figure 4 shows the plot generated by the above procedure. It is important to note that we are plotting $\mu'$ from the formula, not $1/\mu'$. At first glance we may wonder why we get such a striking pattern of ellipses, each comprised of dots of the same colour. The first thing to notice is the semi-ellipses that surround the origin. This is because we have restricted our range of $c$ and $d$ to a finite size. If we were to compute the infinite set of values we would see these circular regions crunching in towards zero, which tells us that the limit point of the set of limit points is zero. Note that when the procedure is executed the plot produced will also contain a point at zero, however this is to be ignored as it is a result of empty entries in the matrix $x$, where $c$ and $d$ are not co-prime. We also see from the plot that if $a$ and $b$ are real then the circular regions are centred around the origin, whereas if $\Im(a) \neq 0$ or $\Im(b) \neq 0$ then we see these regions being twisted and contorted around the origin.

The reason for the elliptic nature of the plots goes much deeper, here we will briefly outline why such ellipses arise. The proof of the theorem that produces the formula (1) above shows that the points $1/\mu'$ lie on a lattice on the complex plane. The vertical axis corresponds values of $c$ and the horizontal axis corresponds to values of $d$. Points on the lattice will be missing whenever $\gcd(c, d) \neq 1$. Now consider the map $\tau : z \to 1/z$ for $z \in \mathbb{C}$. This map takes $0 \to \infty$, $\infty \to 0$ and fixes 1 and $-1$. So essentially it maps lines to circles. So when we map the arithmetic-geometric mean given by (1) we are mapping each vertical

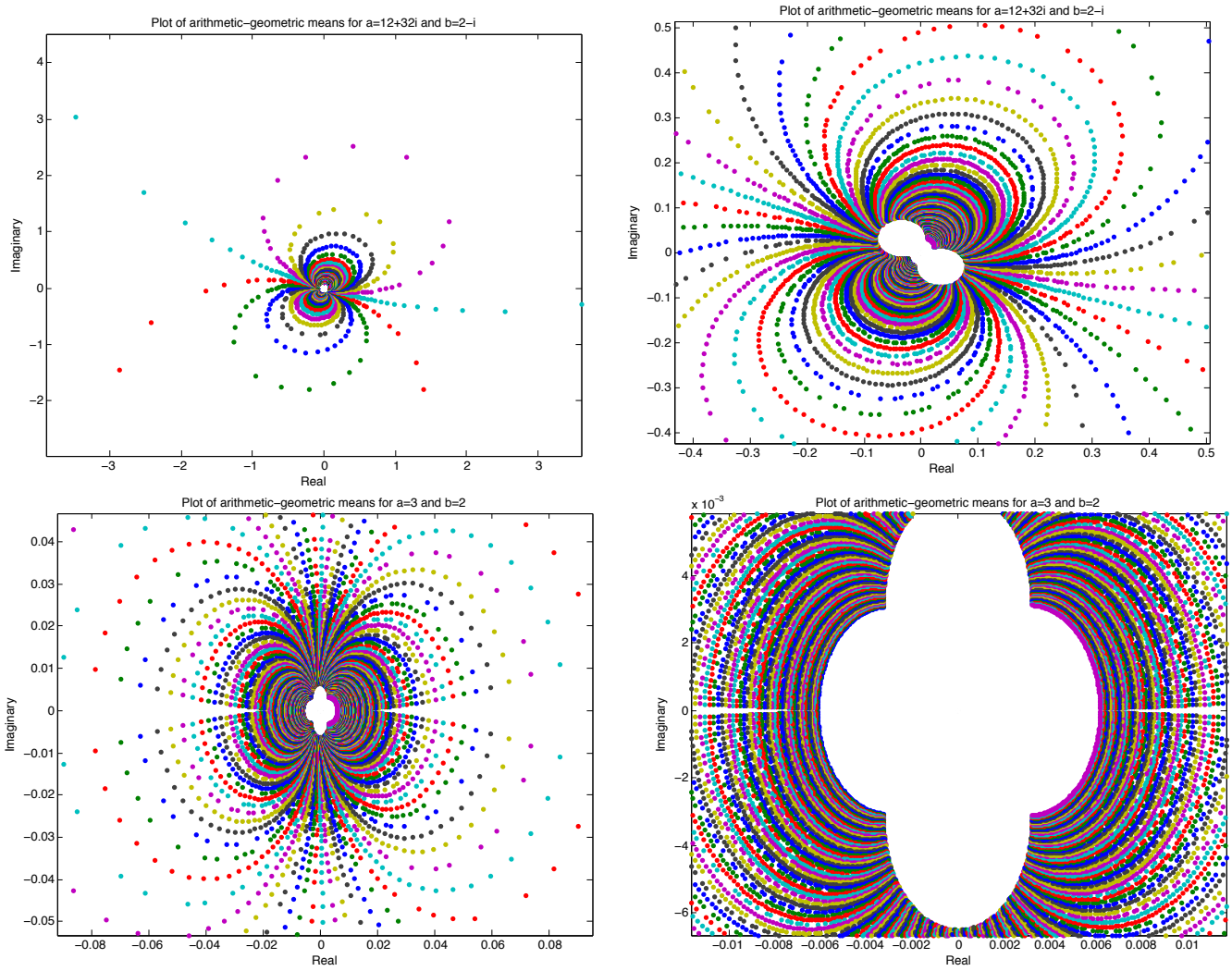



Figure 4: Arithmetic-geometric means of $a = 12 + 32i$ and $b = 2 - i$ at various resolutions (top) and a plot of the arithmetic-geometric mean of two real numbers (bottom).

line of the lattice to an ellipse. This means that the colour of each ellipse corresponds to a particular value of $d$ and the number of points on each ellipse is determined by the range of values of $c$.

We would like to observe the trajectories of every possible sequence for a finite number of iterations after which point we would like to see where these sequences converge to. So we will iterate the sequence $N$ times such that the wrong (i.e. not right) choice could be made in the first $0, .., n$ steps, where $n < N$. In order to do this we construct two $(n+1)$x$2^n$ matrices $x$ and $y$ in which every column will be a different possible trajectory of sequences generated by specified $a, b$. We will first consider the matrix $y$. This will contain simply $b$ for every entry in the first row, corresponding to the fact that there is no choice of values for our initial $b$. Each entry in the $k$th position of every $j$th row of $y$ will correspond to the geometric mean of $x_{(j-1,k)}$ and $y_{(j-1,k)}$. However as MATLAB always returns the positive square root of a number we need a way to generate every possible value for $b_{(j,k)}$.

Every $j$th row should contain $2^{j-1}$ possible values for $b_{(j,k)}$ so we partition each row into $2^{n-(j-1)}$ parts. We then index the partitions from 1 to $2^{(j-1)}$. We then run across the entries of the row and take the positive root if the index of the partition is odd, and the negative square root if the index is even. The resulting matrix will have every possible trajectory after $n$ iterations. Once we have generated the $j$th row of the $y$ matrix we simply run a loop over the length of a row in $x$ and calculate $x_{(j,k)} = (x_{(j-1,k)} + y_{(j-1,k)})/2$ We then call the function agmsimp.m which iterates the entries in the last row of $x$ and $y$, making the right choice for $n_1 = N - n$ iterations. The matrices generated by looping the function agmsimp.m over these rows are then concatenated with the matrices $x$ and $y$. We do this by calling the procedure 'matr.m':

```
a=12+32*1i; %Sets value of a.
b=2-1i; %Sets value of b.
n=5; %Sets the number of iterations for all possible paths of a_n and b_n.
n1=6; %Sets the number of iterations for 'agmsimp' function.

x=zeros(n+1,2^n); %Generates matrix for efficiency.
y=zeros(n+1,2^n); %Generates matrix for efficiency.
t0=zeros(n+n1+1,2^n); %Generates matrix for efficiency.

for i=1:2^n %Loops over columns of x and y.
    x(1,i)=a; %Sets a as every entry in first row of x.
    y(1,i)=b; %Sets b as every entry in first row of y.
end

for j=1:n %Runs over rows of x and y.
    r=length(y)/(2^j); %Calculates number of partitions of each row.
    for k=1:2^j %Runs over the entries in each partition of jth row.
```

```
        s=(k-1)*r+1; %This is the value of the leftmost entry of partition.
        for m=0:r-1 %This runs over each entry in the partition.
            y(j+1,s+m)=sqrt(y(j,s+m)*x(j,s+m)); %Sets geometric mean of
            %x(j,s+m) and y(j,s+m) as y(j+1,s+m) entry of y.
            if rem(k,2)==0
                y(j+1,s+m)=-y(j+1,s+m); %Changes entry of y(j+1,s+m) to
                %-y(j+1,s+m) depending on parity of k.
            end
        end
    end
    for p=1:length(x) %Runs over columns of x.
        x(j+1,p)=(x(j,p)+y(j,p))/2; %Sets arithmetic mean of x(j,p) and
        %y(j,p) as (j+1,p)-entry of x.
    end

end

simpx=zeros(n1,2^n); %Generates zero matrix for efficiency.
simpy=zeros(n1,2^n); %Generates zero matrix for efficiency.

for j=1:length(x) %Runs over columns of x.
    [xm, ym]=agmsimp(x(n+1,j),y(n+1,j),n1+1); %Calculates simplest value
    %sequence of x(n+1,j) and y(n+1,j) entries for n1+1 iterations.
    simpx(1:n1,j)=xm(2:n1+1); %Sets vector xm as jth column of simpx.
    simpy(1:n1,j)=ym(2:n1+1); %Sets vector ym as jth column of simpy.
end

finx=[x;simpx]; %Concatenates matrix x with matrix simpx.
finy=[y;simpy]; %Concatenates matrix y with matrix simpy.

for j=1:length(x) %These loops generate a matrix of the same size as finx.
    for k=1:n+n1+1 %and finy where each column counts the total number of
        t0(k,j)=k-1; %iterations of the above process for plotting.
    end
end
```

Once we have this matrix we can combine it with the values generated in the procedure 'means.m' to analyse where each trajectory ends up in the complex plane. We will first consider the magnitude of $a_n$ and $b_n$ at each iteration by taking the absolute value of each entry in 'finx' and 'finy'. We first call the procedure 'matr.m', followed by 'means.m' and we then execute the following command:

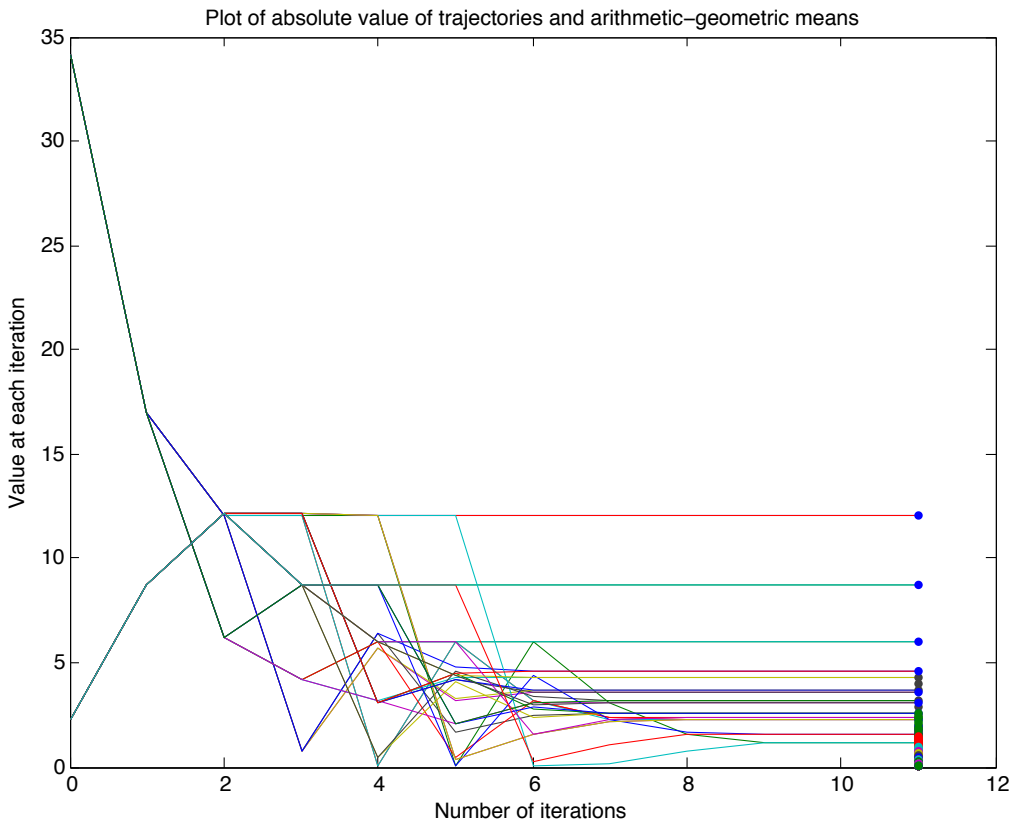



Figure 5

```
>> plot(t0,abs(finx),t0,abs(finy),t13,abs(x),'.')
```

This produces the plot in Figure 5. The length of the horizontal line is an indicator of how many "wrong" choices we have made for $b_n$: the longer the horizontal line segment the more right choices we have made. The dots at $n = 11$ indicate the absolute value of the arithmetic-geometric means calculated using the procedure 'means.m'. What we observe straight away is that the number of right choices has a direct impact on the magnitude of the common limit of the sequences. It seems that the more right choices we make for each $b_n$, the larger the size of the arithmetic-geometric mean. This is interesting, but there is more that we can do with these trajectories. To really understand how these sequences evolve as we iterate we can plot a 3-dimensional trajectory by executing the command

```
>> plot3(t0,real(finx),imag(finx),t0,real(finy),imag(finy),
        t13,real(x),imag(x),'.')
```

but this is not helpful when reproduced in a document, so it is attached as an interactive file (trajectories.fig). What we see from this trajectory is that there is a correspondence between the number of possible right or "wrong" choices we make and the ellipse that each trajectory ends up on. To explore this more fully we can take the very last row of our matrix 'finx' and experiment with our ranges of $c$ and $d$ to find which values correspond to the number of potential wrong choices we could make in our sequence.

After some experimentation we find that if we were to make the wrong choice for $b_1$ after just one iteration, then there are $2^1$ possible arithmetic-geometric means and they correspond to when $c \in \{-4, 0\}$ and $d = 1$. If we were to make the wrong choice in up to the first 2 steps of the sequence and then the right choice for $n \geqslant 3$ then we have $2^2$ possible arithmetic-geometric means corresponding to when $c \in \{-8, -4, 0, 4\}$ and $d = 1$. If we were to make the wrong choice in up to the first 3 steps then the possible means correspond to when $c \in \{-8, -4, 0, 4\}$ and $d = 1$ as before, but we also have all the means possible if we do not make the right choice in the third step which correspond to when $d = -3$ and $c \in \{-4, 4, 8, 16\}$. We can plot these by setting $c$ as a vector corresponding to the values listed in the procedure 'means.m' and calling 'matr.m' for $n = 1, 2, 3, ....$ The results can be seen in Figure 6. This is really interesting as it means that we can partition all the possible values that equation (1) can produce according to the number of wrong choices we have made at each step in the sequence. This is the same as saying that the number of wrong choices we make is directly related to our values for $c$ and $d$.

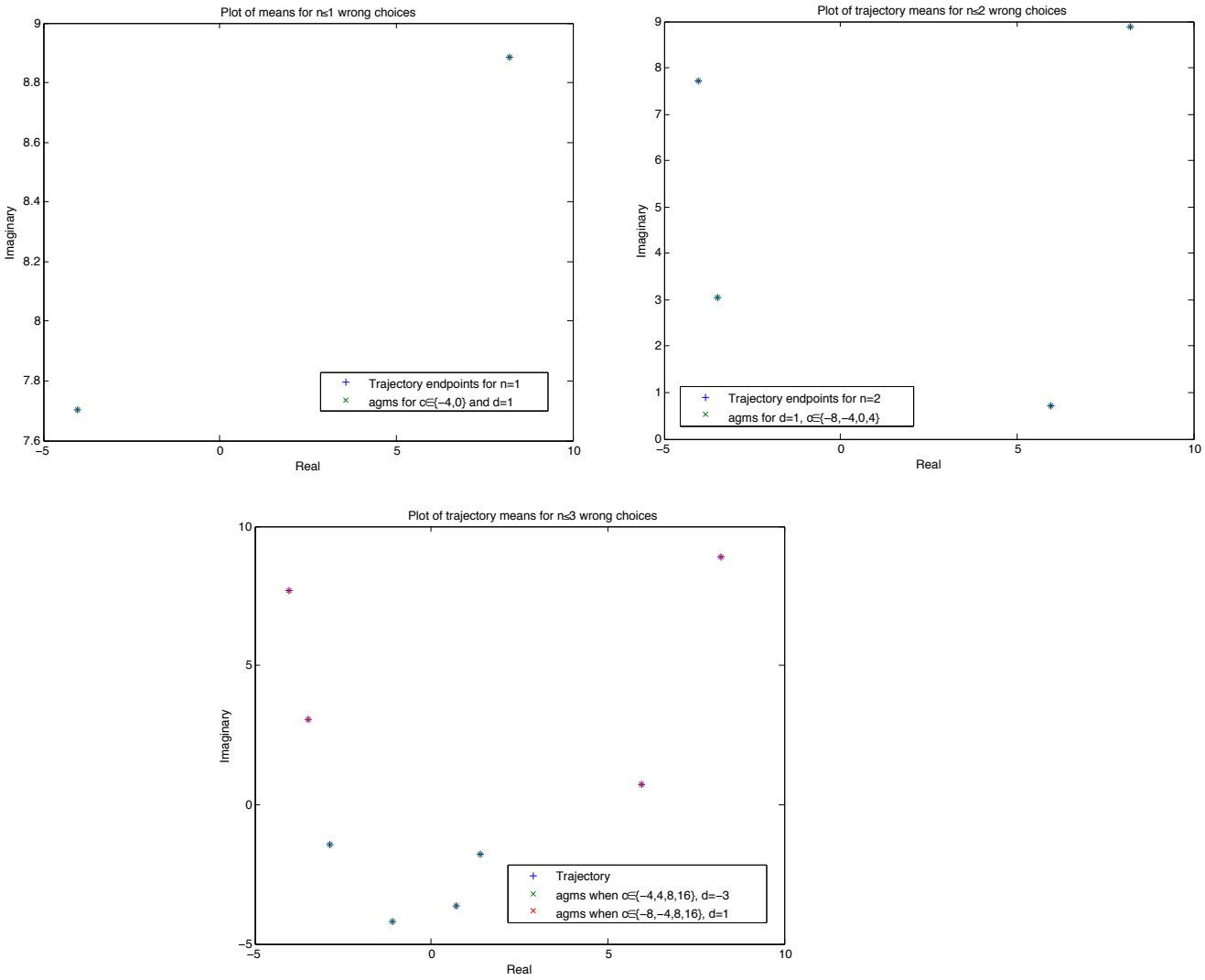

Plot of means for n≤1 wrong choices
Imaginary
Real
Trajectory endpoints for n=1
agms for c∈{−4,0} and d=1
Plot of trajectory means for n≤2 wrong choices
Imaginary
Real
Trajectory endpoints for n=2
agms for d=1, c∈{−8,−4,0,4}
Plot of trajectory means for n≤3 wrong choices
Imaginary
Real
Trajectory
agms when c∈{−4,4,8,16}, d=−3
agms when c∈{−8,−4,8,16}, d=1

Figure 6

# 4 Conclusion

We began by considering the arithmetic-geometric mean of real numbers. We saw how quickly two sequences generated by this simple algorithm will converge. We observed that it is this rate of convergence that makes the arithmetic-geometric mean such an effective tool for calculating $\pi$. Despite the limitations of programming in MATLAB, it should be clear how the arithmetic-geometric mean is a much more efficient method for calculating $\pi$ to very high levels of accuracy compared to Viete's formula. For more information on how the arithmetic-geometric mean can be used to calculate other transcendental numbers and also for thorough discussions of the complexity of such algorithms one may consult the Borwein brothers' book- *Pi and the AGM* (Wiley-Interscience, 1987). It turns out that the arithmetic-geometric mean is deeply linked with elliptic integrals of the first and second kind, so an area that may be interesting to explore would be how we can use this mean to compute the arc length of any given ellipse.

We then explored computationally how the mean behaves on the complex plane. We first analysed how the different means for given $a$, $b$ are distributed on the complex plane and linked this to the theory behind the formula. We then produced some code to generate the possible trajectories of a pair of sequences for a finite number of potential choices of $b_n$. Once these trajectories had been obtained it was particularly interesting to see how the number of potentially wrong choices corresponded to sets of specific values of $c$ and $d$ in our given formula. This is something that would be worth exploring further, as it could lead to a completely different way of classifying and defining these good sequences and their corresponding common limits. Potentially we could separate these common limits into equivalence classes according to how many wrong choices we make, i.e according to the values $c$ and $d$ that generate them.