

Su Nguyen  
**Ways of ESTIMATING  $\pi$**   
Math 107B  
June 14, 2002

The number  $\pi$  has been the subject of a great deal of mathematical (and popular) folklore.

It's been worshipped, maligned, and misunderstood.

$\pi$  has recently (Sep. 20, 1999) been computed to a world record 206,158,430,208 decimal digits ( $2^{36}$ ).

There are many algorithms for **estimating**  $\pi$ , ranging from the Monte Carlo (by generating random points and testing whether or not they fall inside the unit circle.  $\pi$  is approximately  $4 * nHits / nTrials$ ) to the Continued Fraction method. The remarkable result is that the probability is directly **related** to the **value of**  $\pi$ .

Since the actual value of  $\pi$  has been calculated to be equal to

3.141592653589793238462643383279502884197.....

(and has been computed using more powerful hardware and software rather than using Maple 7),

the purpose of this assignment is **NOT to calculate the actual value of  $\pi$** .

Therefore, the main **PURPOSES** of this assignment are:

- (1) Combine the uses and capabilities of Maple 7 and mathematics to estimate  $\pi$ .
- (2) Use several different methods (ranging from the least accurate to the more precise ones) to **estimate** the number  $\pi$ .
- (3) Involve Maple programming and graphics tools to better visualize the materials presented.
- (4) Maple will output the differences between the estimated value of  $\pi$  and the actual value of  $\pi$  itself.

### Calculating $\pi$ using Circumference and Diameter

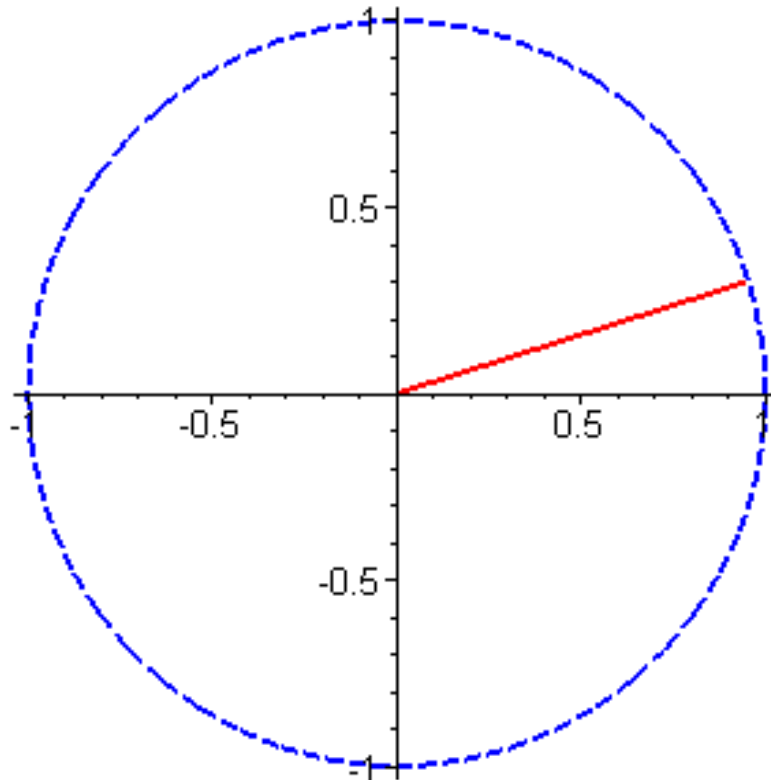
$\pi$  is defined as the ratio of a circle's circumference  $C$  to its diameter  $d = 2r$ .

$$\pi = C / d = C / 2r.$$

$\pi$  is equal to 3.141592653589793238462643383279502884197.....

```
> restart;
with(plottools):
c := circle([0,0], 1, color=blue, linestyle=3,
thickness=2):
j := line([0,0], [.95, .3], color=red, linestyle=1,
thickness=2):
```

```
plots[display](c,j,scaling=CONSTRAINED);
```



In this case, we assume that a circle has a radius of 1 unit, so the diameter is  $2*r = 2*1 = 2$

```
#Display the integration function from 0 to 1 to calculate the Circumference
```

```
C := Int( 4 * ( 1 /sqrt(1-x^2) ), x=0..1 );
```

$$C := \int_0^1 4 \frac{1}{\sqrt{1-x^2}} dx$$

```
> #Evaluate the integral from 0 to 1
```

```
C := int( 4 * ( 1 / sqrt(1-x^2) ), x=0..1 );
```

```
C := 2 π
```

```
> #Pi is equal to the circumference/diameter, displayed 100 digits
```

```
evalf(C/2, 100);
```

```
3.14159265358979323846264338327950288419716939937510582097494459230781
6406286208998628034825342117068
```

```
>
```

Estimating PI using simple methods



```
#Display the difference between calculated value and Pi
evalf( Pi-P );
```

$$P := \frac{1}{6} \ln(2198) \sqrt{6}$$

$$P := 3.1415943494500818302$$

$$-.16958602885917 \cdot 10^{-5}$$

```
> #A ln and sqrt number that is close to Pi
P := ln(2625374126407687744) / sqrt(163);
P := evalf(P);
```

```
#Display the difference between calculated value and Pi
evalf( Pi-P );
```

$$P := \frac{1}{163} \ln(2625374126407687744) \sqrt{163}$$

$$P := 3.3219450371972011299$$

$$-.1803523836074078914$$

```
> #Using equations to calculated a value that is close to Pi
a := 0:
b := 1:
c := 1 / sqrt(2):
d := 1 / 4:
e := 1:
a := b:
b := (b + c) / 2:
c := sqrt(c * a):
d := d - (e * (b - a) * (b - a)):
e := 2 * e:
f := (b * b) / d:
g := ((b + c) * (b + c)) / (4 * d);
g := evalf(g);
```

```
#Display the difference between calculated value and Pi
evalf( abs(Pi-g) );
```

$$g := \frac{1}{4} \frac{\left(\frac{1}{2} + \frac{1}{4} \sqrt{2} + \frac{1}{2} 2^{(3/4)}\right)^2}{\frac{1}{4} - \left(-\frac{1}{2} + \frac{1}{4} \sqrt{2}\right)^2}$$

$$g := 3.1405792505221682482$$

$$.0010134030676249903$$

```
>
```

## Estimating PI by Archimedes Algorithm

This section will use Archimedes algorithm to provide successive approximations to  $\pi$ .

Archimedes obtained the first rigorous approximation of  $\pi$  by inscribing on a circle.

As shown in the graph below, if we divide the the circle into 10 triangles, each angle theta of the triangle is 36 degrees.

We use the formula:  $\tan^{-1}(36) = 1/x$ , where x is the opposite side of the angle theta (36 degrees).

If we sum up all the x values of the 10 triangles, we would get the approximation to the circumference of the circle.

Then  $\pi$  is **approximated** by the formula:  $\pi = C/d = C/2r$ .

```
> #Graph the example of a circle of radius 1 circumscribed  
an octagon.
```

```
restart;
```

```
with(plots):
```

```
with(plottools):
```

```
one_poly := [[0,0],[0,1],[1,1],[1,0]]:
```

```
polygonplot(one_poly,axes=boxed):
```

```
e := circle([0,0], 1, color = blue, linestyle=3,  
thickness=1):
```

```
f := line([0,0], [-.31,.95], color=red, linestyle=1,  
thickness=2):
```

```
g := line([0,0], [.31,.95], color=red, linestyle=1,  
thickness=2):
```

```
h := line([0,0], [.81,.58], color=red, linestyle=1,  
thickness=2):
```

```
i := line([0,0], [1,0], color=red, linestyle=1,  
thickness=2):
```

```
ngon := n -> [seq([ cos(2*Pi*i/n), sin(2*Pi*i/n) ], i =  
1..n)]:
```

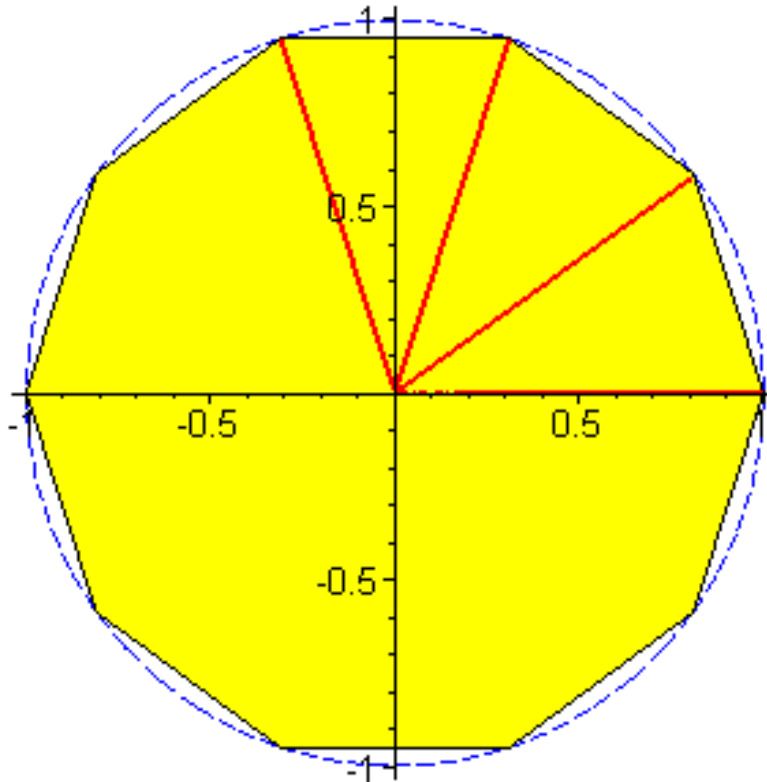
```
display( ([ polygonplot(ngon(10)),
```

```
textplot([0,0,`Octagon`] ) ],
```

```
color=yellow),e,f,g,h,i,scaling=CONSTRAINED );
```

```
Warning, the name changecoords has been redefined
```

```
Warning, the name arrow has been redefined
```



```

> EstPI1 := proc(n)
  #Declare and initialize local variables
  local c, num;
  #Find the artan of (360 degrees/n triangles)
  c := evalf( arctan(Pi/n) );
  #Num equal to (c * n triangles)
  num := evalf(c*n);
  #Return the calculated value of Pi
  return num;
end proc;

#Assume there are 1000 to 1010 triangles inscribed on a
circle
for i from 1000 to 1010 do
  #Call procedure EstPI1 with i between 1000 and 1010
  EstPI1(i);
end do;

```

3.141582319

3.141582340

3.141582359

3.141582380

3.141582402

3.141582421

3.141582441

3.141582462

3.141582482

3.141582502

3.141582522

```
> #Display the difference between calculated value and Pi
evalf( Pi - EstPI1(1010) );
.000010132
```

>

## Estimating PI using Monte Carlo Method

This section will use probability theory to estimate the number  $\pi$ , the area of a circle with unit radius.

This method produce an approximation of  $\pi$  by generating random points and testing whether or not those random points fall inside the unit circle.  $\pi$  is approximately  $4*nHits / nTrials$ .

The program will calculate the distance from the circle center to coordinates (x,y) to determine

whether (x,y) is inside the circle or not. It counts the total points land inside the circle and divided by the number of iterations.

The result will be the **approximation** to  $\pi$ .

```
> restart;
> #Include the plots packages
with(plots):
Warning, the name changecoords has been redefined

> #Setting up the graph ready to be plotted
PIPix := proc(pts::list, EstPie::float)
  #Declare local variables to be used in this section
  local CirclePlot, Plot1, Plot2, PTitle, PlotPoint;
  PTitle := cat("The Estimate of PI is: ", convert(EstPie,
string));
  CirclePlot := plot({sqrt(1-x*x), -1*sqrt(1-x*x)}, x=-
1..1, color=green,
                    title = PTitle);
  Plot1 := plot([[0,1], [1,1]]);
  Plot2 := plot([[1,1], [1,0]]);
  PlotPoint := plot(pts, color = blue);
  display( {CirclePlot, Plot1, Plot2, PlotPoint},
scaling=CONSTRAINED );
end proc;
```

```

> RandNum := proc()
  #Generate and return a random number between 0 and 1
  return evalf(rand() / 10^12);
end proc:

> DrawPi := proc(Points::integer)
  local x, y, i, pts, Circ, pi, CirclePlot, PlotPoint,
Plot1, Plot2 ;
  pts := [] ; Circ := 0 ;

  #Begin of for loop from 1 to Points
  for i from 1 to Points do
    x := RandNum() ; y := RandNum() ;
    pts := [op(pts), [x,y]] ;
    if ( (x*x)+(y*y) ) <= 1 then Circ := Circ + 1 ; end if
  ;
  end do ;

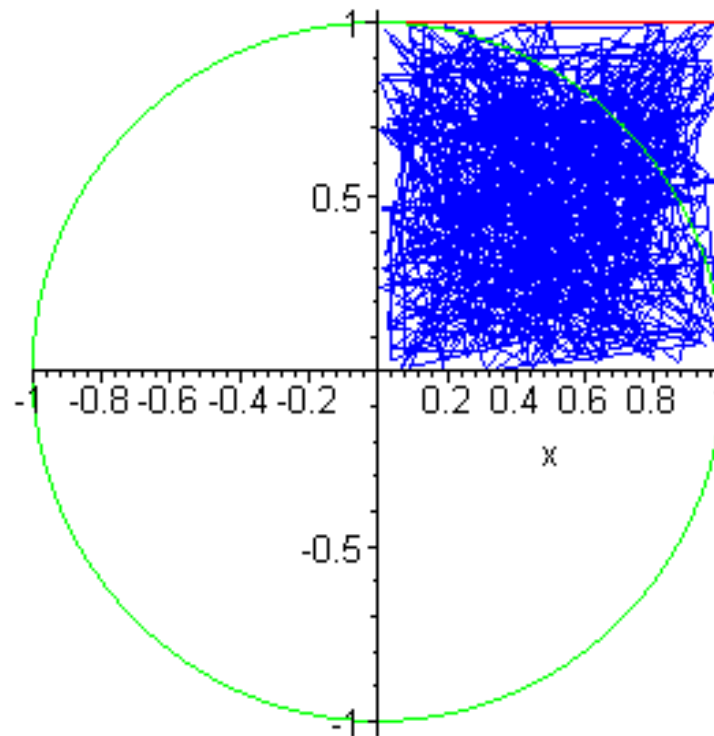
  #Evaluate the value pf pi
  pi := evalf(4 * Circ / Points) ;

  PIPix(pts, pi);
end proc:

> #Estimated the value of Pi at 500 points
DrawPi(500);

```

The Estimate of PI is: 3.120000000



>

## Estimating PI by adding up the first MAX terms of the series

This section will add up the first MAX terms of the series to **approximate**  $\pi$ . Since the series is alternating, the result alternately gives the upper and lower estimate to  $\pi$ .

$$\pi = 4 - (4/3) + (4/5) - (4/7) + \dots - (-1)^n [4/(2n+1)] + \dots,$$

Noted that the sign changes from positive to negative or vice versa and the denominator is incremented by 2.

```
> EstPI2 := proc(n)
#Declare and initialize local variables
local sgn, dnm, term, sm, count;
sgn := 1:
dnm := 1:
term := 0:
sm := 0.0:

#Begin for loop from 0 to n, the number of iterations
for count from 0 to n do
  #Denominator equal 2*count + 1
  dnm := 2*count + 1:
  #Term equal sign (+ or -) * 4/denominator
  term := sgn * (4.0/dnm):
```

```

    #Sum = sum + term
    sm := sm + term:
    #Switch the sign after each term
    sgn := -sgn:
end do;

#Return the sum, which is the caculated value of Pi
return sm;
end proc:

#Assume i is from 100 to 105 iterations
for i from 10000 to 10005 do
    #Calculated the value of Pi from 100 to 105 iterations
    EstPI2(i);
end do;

3.141692612
3.141492642
3.141692592
3.141492662
3.141692572
3.141492682

> #Display the difference between the calculated value and
Pi.
evalf ( Pi - EstPI2(105) );
.009433752

```

>

## Estimating PI using the Double Exponential Random Variables Method

This section will simulate Double Exponential Random Variables.

First, we calculate the cdf (cummulative distribution function) and inverse cdf of the Double Exponential distribution  $0.5e^{-0.5(x^2+y^2)}$  dx dy, and then generate uniform random variables

and plug them in. It will generate pairs of Double Exponential random variables

and these are plugged into  $2*\exp(-.5*((x^2)+(y^2))+|x|+|y|)$

as x and y arguments of the function. Finally, it will

calculate the average over all the iterations as an **approximation** of  $\pi$ .

```

> RandNum := proc()
    #Generates a random number between 0 and 1
    return evalf(rand() / 10^12);
end proc:

```

```

> EstPI4 := proc(n)
#Declare and initialize local variables
local i, sm, a, b, x, y, doubleexp;
sm := 0;

#Begin for loop from 1 to n, the number of iterations
for i from 1 to n do
  #Assign a random number to variables a and b
  a := RandNum();
  b := RandNum();

  #Using if then else statement for appropriate calculation
  if a <= .5 then x := log(2*a); else x := -log(2-2*a); end
if;
  if b <= .5 then y := log(2*b); else y := -log(2-2*b); end
if;
  doubleexp := evalf( 2 * exp( (-.5 * ((x*x) + (y*y))) +
abs(x) + abs(y) ) );
  sm := evalf(doubleexp + sm);
end do;
#End of the for loop

#Return sum / n, which is the calculated value of Pi
return (sm/n);
end proc;
#End the procedure

> #Assume the number of iterations is 1000
EstPI4(1000);
                                     3.126578596

> #Display the difference between the caculated value and Pi
evalf( Pi - EstPI4(1000) );
                                     -.005147429

>

```

## Estimating PI using Continued Fraction

This section use the Continued Fraction method to estimate  $\pi$ . We assumed that the value of  $\pi$  is known (3.1415...), then we can approximate  $\pi$  using the continued fraction method.

$$\pi = a_1 + 1 / ( a_2 + 1 / ( a_3 + 1 / ( a_4 + 1 / a_5 + \dots ) ) )$$

$$\pi = 3 + 1 / ( 7 + 1 / ( 15 + 1 / ( 1 + 1 / 292 + \dots ) ) )$$

Continued fraction provide a series of "best" estimates for an irrational number.

The "best" approximation of a given order, is

[3,7,15,1,292,1,1,1,2,1,3,1,14,2,1,1,2,2,2,...]

The very large term 292 means that the convergent:  $[3,7,15,1] = [3,7,16] = 355/113 = 3.14159292\dots$

is an extremely good approximation. The first few convergents are 22/7, 333/106, 355/113, 103993/33102, 104348/33215, ...

A nice expression for the third convergent of  $\pi$  is given by:

$$\pi = 2[1,1,1,3,32] = 355/113 = 3.14159292\dots$$

```
> #Generate the "best" approximation of a given order
GenerateNum := proc (x, n)
#Declare and initialize local variables
local i, num, a;
num := evalf(x, 1000);
a := [seq(0, i=1..n)];

#Begin the for loop from 1 to n
for i from 1 to n do
  #Take the floor of num and insert into array a of element
  i
  a[i] := floor(num);
  num := evalf (1 / (num - a[i]), 1000);
end do;
#End of for loop
#Return the array a
return a;

end proc;
#End of procedure

> #Give an example of the "best" approximation of a given
order
GenerateNum(Pi, 10);
#Assign array d to store the numbers generated by
GenerateNum
d := GenerateNum(Pi, 10);
#Display the size of array d
nops(d);
#Display the 3rd element stored in array d
d[3];
#Display up to the 10th fractions that was calculated to Pi
convergs( d(10) );
```

[3, 7, 15, 1, 292, 1, 1, 1, 2, 1]

d := [3, 7, 15, 1, 292, 1, 1, 1, 2, 1]

10

15

$$2, \frac{22}{7}$$

$$3, \frac{333}{106}$$

$$4, \frac{355}{113}$$

$$5, \frac{103993}{33102}$$

$$6, \frac{104348}{33215}$$

$$7, \frac{208341}{66317}$$

$$8, \frac{312689}{99532}$$

$$9, \frac{833719}{265381}$$

$$10, \frac{1146408}{364913}$$

```

> EstPI5 := proc (n)
#Declare and initialize local variables
local aysize, b, d, num;
num := 0;
#Assign array d to store the numbers generated by
GenerateNum
d := GenerateNum(Pi, n);
#Assign aysize to store the size of array d
aysize := nops(d);

#Begin for loop from aysize to 2 by decrease -1 each time
for b from aysize by -1 to 2 do
    num := evalf( 1 / (d[b] + num) , 1000);
end do;
#End of for loop

#Return the calculated value of Pi
return (3+num);
end proc;
#End of procedure

> #Set the iterations to 75 times, display 100 digits
evalf( EstPI5(75), 100 );

```

3.14159265358979323846264338327950288419716939937510582097494459230781  
6406275163263067827826427951239

> #The continued fraction method proved to be the most  
accurate

#method of all of the above methods as a way get an  
approximation to Pi.

evalf( (Pi - EstPI5(75)), 100 );

.11045735560206998914165829 10<sup>-73</sup>

>

>