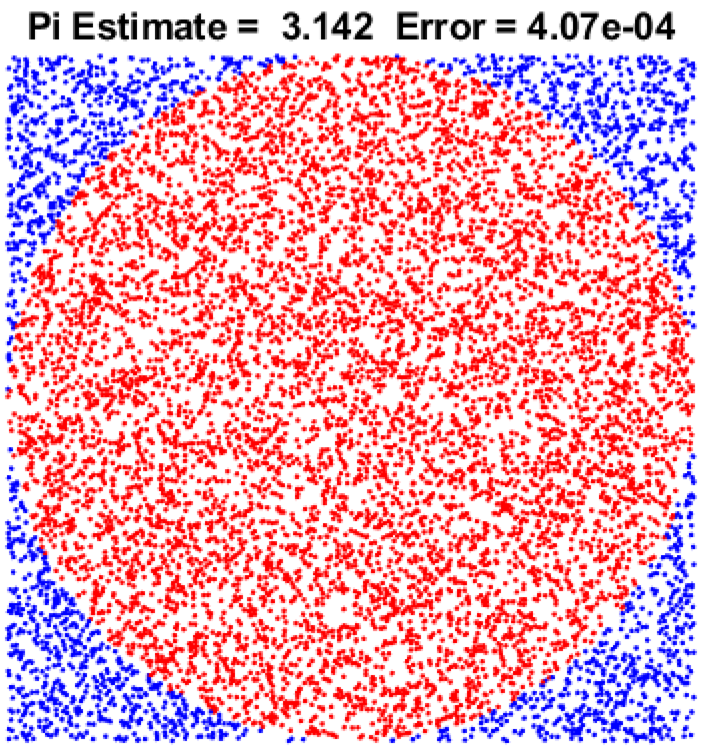
- Previous Lecture:
  - (Definite) iteration using `for`
- Today's Lecture:
  - Review loop & conditionals using graphics (I)
  - (Indefinite) iteration using `while`
- Announcements:
  - Please fill out "Week 3 Survey" in CMS
  - Be sure to read *Insight* §3.2 before discussion section next week
  - 1-on-1 tutoring is available via CMS
    - Office and consulting hours also available to help you – let us clarify anything that doesn't make sense
  - Project 2 (part A) will be posted before the weekend
  - (if you already know another language) We do not use `break` in this course

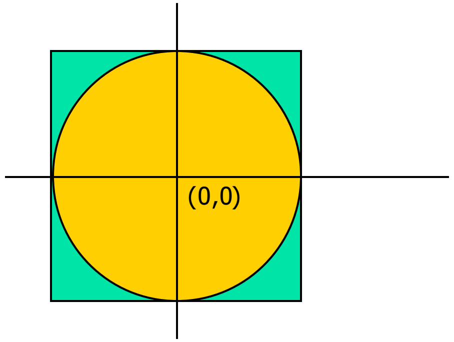
Monte Carlo π with N darts on L-by-L board

- Be output-oriented
  - Want a square full of random darts
  - Want to treat darts in a circle specially
- Outline steps to produce desired output (which should be repeated?)
  - *"Throw" dart to random location*
  - *Determine whether dart is in circle*
- Make implementation decisions (*after* writing down outline)
  - Coordinate system? Origin?
  - Circle test?
- Compare output with expectations



Monte Carlo π with N darts on L-by-L board

```
N=__; L=__; hits= 0;
for k = 1:N
    % Throw kth dart
    x= rand()*L – L/2;
    y= rand()*L – L/2;
    % Count it if it is in the circle
    if sqrt(x^2 + y^2) <= L/2
        hits= hits + 1;
    end
end
myPi= 4*hits/N;
```



Definite iteration

Accumulation

Visualize output (check your own work!)

If dart is inside circle

Draw red dot

Otherwise

Draw blue dot

Graphics details

- `hold on`, `hold off`
  - Add to existing plot, or replace?
- `axis equal`, `axis off`, `axis()`
  - For graphics, want square aspect ratio, no distracting tic marks
  - Manual control of range
- `sprintf()`
  - Insert numbers into text variables

What will be displayed when you run the following script?

```
for k = 4:5
    disp(k)
    k= 9;
    disp(k)
end
```

A
4
9
5
9

B
4
4
5
5

C
4
9

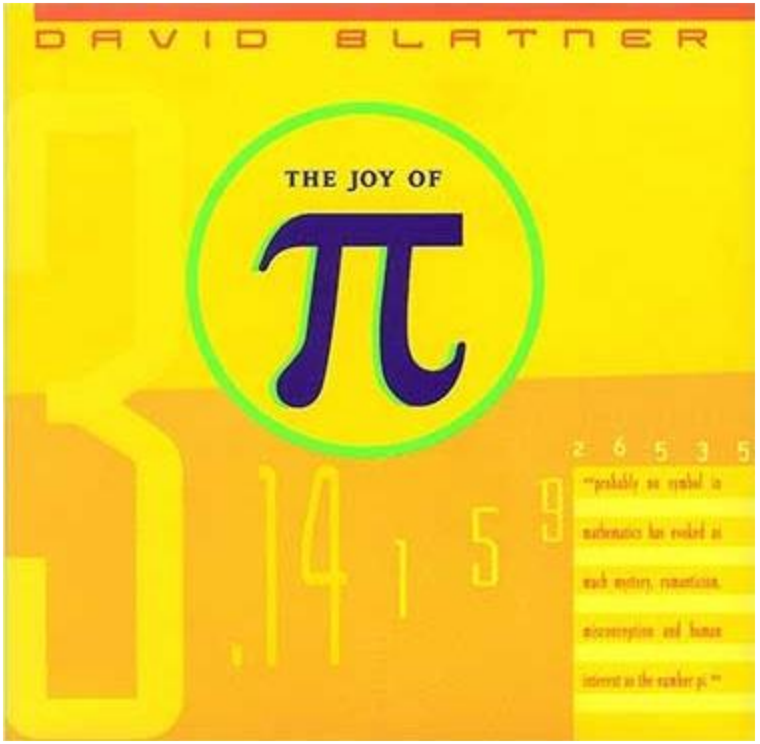
D
error

Watch MatTV to learn more!

Episode IX:
Troubleshooting Loops

Approximating π

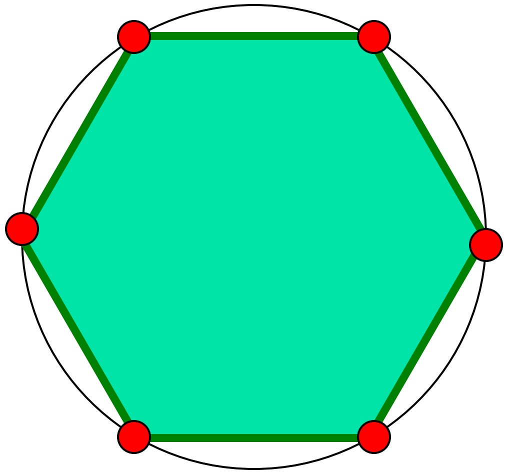
- Why?
  - Today's convenience made possible *because of* computers
- Methods
  - Monte Carlo
  - Series summation (exercise 3)
  - Polygons (Ch. 2)
  - Fractions (Ch. 3)
- Properties of approximations
  - Speed of convergence
  - **Error bounds**

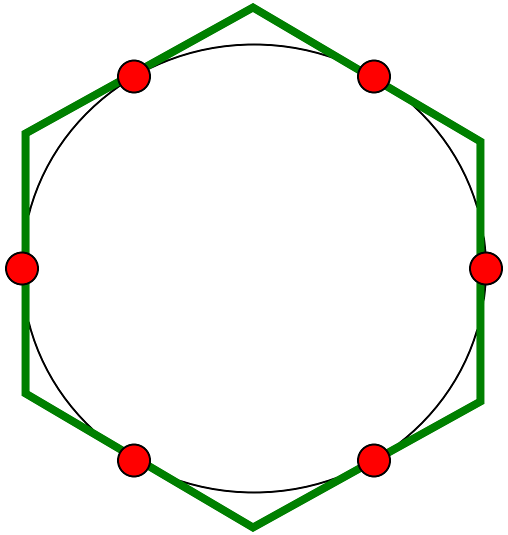


$$T_n = 1 + \frac{1}{2^2} + \cdots + \frac{1}{n^2} = \sum_{k=1}^{n} \frac{1}{k^2} \approx \frac{\pi^2}{6}$$

$$R_n = 1 - \frac{1}{3} + \cdots + \frac{(-1)^{n+1}}{2n-1} = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1} \approx \frac{\pi}{4}$$

Example: n-gon → circle





Inscribed hexagon

$(n/2)\sin(2\pi/n)$

Circumscribed hexagon

$n\tan(\pi/n)$

As *n* approaches infinity, the inscribed and circumscribed areas approach the area of a circle.

When will |OuterA – InnerA| <= .000001?

Outline

- *Input tolerance*
- *Compute areas of inscribed and circumscribed triangles*
- *Compute difference in areas*
- *Repeat until difference is smaller than tolerance:*
  - *Compute areas of inscribed and circumscribed polygons with one more side*
  - *Compute difference in areas*
- *Output number of sides, average area, and difference*

Can we do this?

- Previously, made decisions while looping
  - Can *nest* conditionals inside of loops
  - But always looped a fixed number of times
- Now, need to make decisions *that affect* looping
  - Need something new

```
tol= input('Enter the error tolerance:');


% The triangle case...
n= 3;                     % Number of Polygon Edges
A_n= (n/2)*sin(2*pi/n);  % Inscribed Area
B_n= n*tan(pi/n);         % Circumscribed Area
ErrorBound= B_n - A_n;    % The error bound

% Repeat until error less than or equal to tolerance
???
    n= n + 1;
    A_n= (n/2)*sin(2*pi/n);
    B_n= n*tan(pi/n);
    ErrorBound= B_n - A_n;
end

% Display the final approximation
fprintf('With %d sides, avg A is %f, diff is  %f\n',
               n,       (A_n+B_n)/2,  ErrorBound);
```

“Until” vs. “As Long As”

Repeat until…

`ErrorBound <= tol`

Stopping condition

Repeat as long as…

- A `ErrorBound <= tol`
- B `ErrorBound <  tol`
- C `ErrorBound >  tol`
- D `ErrorBound >= tol`

Keep-going condition

```matlab
tol= input('Enter the error tolerance:');


% The triangle case...
n= 3;                       % Number of Polygon Edges
A_n= (n/2)*sin(2*pi/n);     % Inscribed Area
B_n= n*tan(pi/n);           % Circumscribed Area
ErrorBound= B_n - A_n;      % The error bound

% Repeat until error less than or equal to tolerance
while ErrorBound > tol
    n= n + 1;
    A_n= (n/2)*sin(2*pi/n);
    B_n= n*tan(pi/n);
    ErrorBound= B_n - A_n;
end

% Display the final approximation
fprintf('With %d sides, avg A is %f, diff is  %f\n',
              n,        (A_n+B_n)/2,  ErrorBound);
```

Iteration caps

- Sometimes dangerous to let computers keep trying to compute something indefinitely
  - "I need to make a decision now; give me your best guess (and how confident you are)"
- *Indefinite* not the same as *infinite*, but infinite becomes a possibility
  - Tip: Ctrl+C to interrupt stuck program
- Common to impose a maximum number of iterations
  - How does our program change?

```
% Approximate pi (from Eg2_2.m)

tol= input('Enter the error tolerance:');
nMax= input('Enter the iteration bound:');

% The triangle case...
n= 3;                        % Number of Polygon Edges
A_n= (n/2)*sin(2*pi/n);      % Inscribed Area
B_n= n*tan(pi/n);            % Circumscribed Area
ErrorBound= B_n - A_n;       % The error bound


% Iterate until error<=delta or until n reaches nMax
while ____________________________________
    n= n + 1;
    A_n= (n/2)*sin(2*pi/n);
    B_n= n*tan(pi/n);
    ErrorBound= B_n - A_n;
end

% Display the final approximation...
```

↑ To-do: Fill in the loop guard (Boolean expression)

```
% Approximate pi (from Eg2_2.m)

tol= input('Enter the error tolerance:');
nMax= input('Enter the iteration bound:');

% The triangle case...
n= 3;                      % Number of Polygon Edges
A_n= (n/2)*sin(2*pi/n);    % Inscribed Area
B_n= n*tan(pi/n);          % Circumscribed Area
ErrorBound= B_n - A_n;     % The error bound


% Iterate until error<=delta or until n reaches nMax
while (ErrorBound > tol  &&  n < nMax)
    n= n + 1;
    A_n= (n/2)*sin(2*pi/n);
    B_n= n*tan(pi/n);
    ErrorBound= B_n - A_n;
end

% Display the final approximation...
```

↑ To-do: Fill in the loop guard (Boolean expression)

Tips: complements and Boolean algebra

- Until *A*
- Until x < y
- Until *A* or *B*
- Until *A* and *B*

```
while ~A   % "not A"
```

```
while ~(x < y)
while x >= y
```

```
while ~(A || B)
while ~A && ~B
```

```
while ~(A && B)
while ~A || ~B
```

Homework exercise: Convince yourselves that these are true

Find smallest *n* such that *outerA* and *innerA* converge

First, itemize the tasks:

- *define how close is close enough*
- *select an initial n*
- *calculate innerA, outerA for current n*
- *diff= outerA – innerA*
- *close enough?*
- *if not, increase n, repeat above tasks*

Find smallest *n* such that *outerA* and *innerA* converge

Now organize the tasks → algorithm:

n gets initial value

innerA, outerA get initial values

Repeat until difference is small:

increase n

calculate innerA, outerA for current n

diff= outerA – innerA

Find smallest *n* such that *outerA* and *innerA* converge

```
n gets initial value
calculate innerA, outerA for current n
while  <difference is not small enough>
    increase  n
    calculate innerA, outerA for current n
    diff= outerA - innerA
end
```

Indefinite iteration

See `Eg2_2.m`

To-do: Modify the script to prompt the user until a delta at least 10^-12 is input

```
tol= input('Enter the error tolerance: ');







n = 3;                           % Number of Polygon Edges
A_n = (n/2)*sin(2*pi/n); % Inscribed Area
B_n = n*tan(pi/n);           % Circumscribed Area
ErrorBound = B_n - A_n;  % The error bound

while (ErrorBound > tol)
    n = n+1;   A_n = (n/2)*sin(2*pi/n);   B_n = n*tan(pi/n);
    ErrorBound = B_n - A_n;
end
% Display the final approximation
```

To-do: Modify the script to prompt the user until a delta at least 10^-12 is input

```
tol= input('Enter the error tolerance: ');
tolMin= 1e-12;
while  tol < tolMin
  tol= input(sprintf('Enter a tolerance >= %.0e: ',tolMin));
end


n = 3;                        % Number of Polygon Edges
A_n = (n/2)*sin(2*pi/n);  % Inscribed Area
B_n = n*tan(pi/n);            % Circumscribed Area
ErrorBound = B_n - A_n;   % The error bound

while (ErrorBound > tol)
    n = n+1;   A_n = (n/2)*sin(2*pi/n);   B_n = n*tan(pi/n);
    ErrorBound = B_n - A_n;
end
% Display the final approximation
```

Important Features of Iteration

- A task can be accomplished if some steps are repeated; these steps form the loop body
- Need a starting point
- Need to know when to stop
- Need to keep track of (and measure) progress

Common loop patterns

Do something *n* times

```
for k= 1:1:n
    % Do something

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something


    % Update loop variables

end
```

Pattern to do something n times

```
for k= 1:1:n
    % Do something

end
```

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

Pattern to do something n times

```
for k= 1:1:n
    % Do something

end
```

```
%Initialize loop variables
k= 1;
while ( k <= n )
    % Do something

    % Update loop variables
    k= k+1;
end
```