

Intro Math Problem Solving

September 21

Factorials

Stepping Stone Sequences

N-Choose-K

Pascal's Triangle

Homework #3

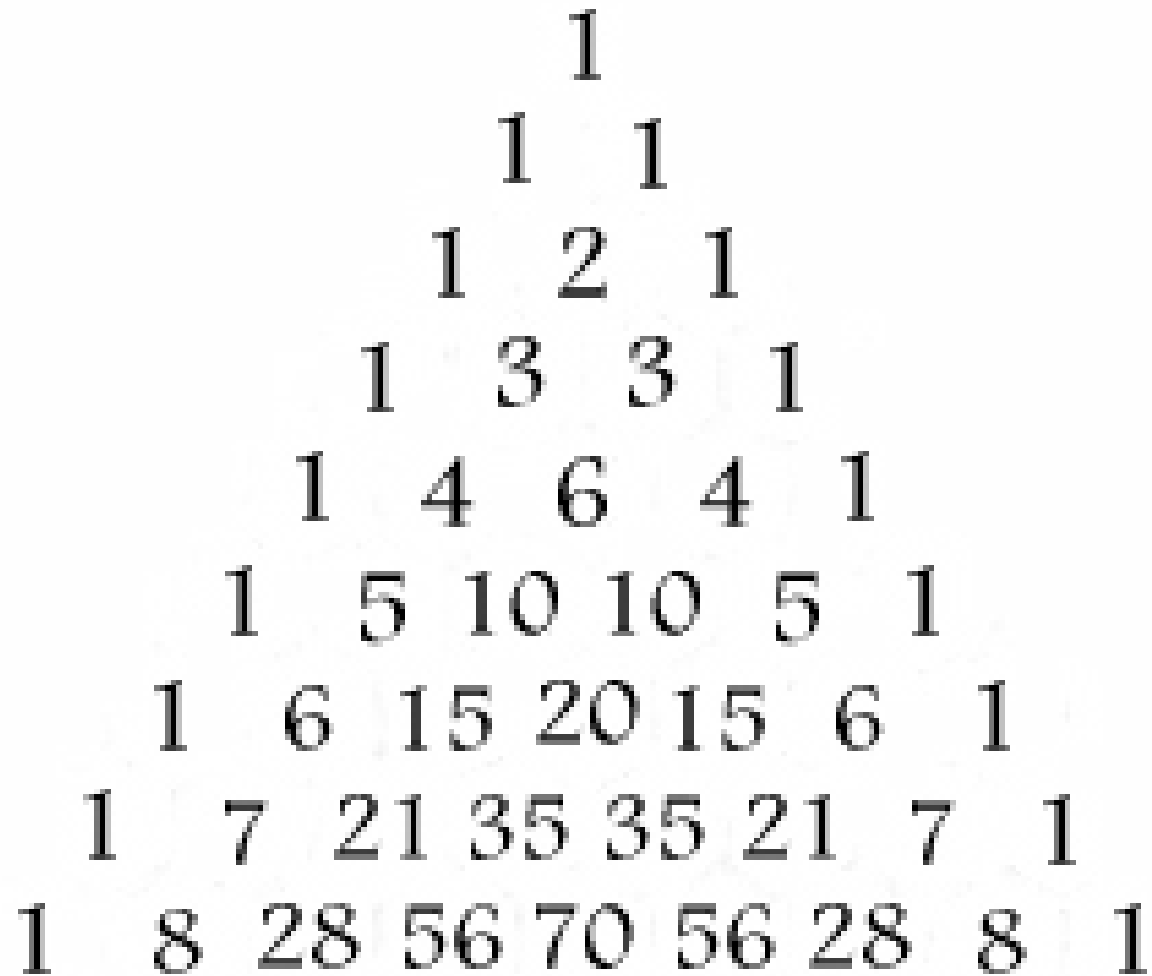
Exam next Thursday

Exercise

Refer to the triangle on the next page.

- 1) What is the next row?
- 2) How does this triangle tell you how to write out $(x+y)^4$?
- 3) In 4 coin tosses, how many ways can I get 2 Heads, 2 Tails?
- 4) What is the probability of 2 Heads, and 2 Tails, in 4 coin tosses?
- 5) How many ways can I choose 5 things from a set of 7?
- 6) Is there a formula for the number in row N, column K?
- 7) Is there a way to write out the N-th row of this triangle, without any other information?
- 8) If a 256 steel balls drop down through a pyramid of nails involving 9 rows, what pattern are they likely to form?

Pascal's Triangle



Pascal's Triangle is a triangular array of binomial coefficients. Each number is the sum of the two numbers directly above it. The triangle is symmetric about its center.

				1																
				1		1														
				1		2		1												
				1		3		3		1										
				1		4		6		4		1								
				1		5		10		10		5		1						
				1		6		15		20		15		6		1				
				1		7		21		35		35		21		7		1		
				1		8		28		56		70		56		28		8		1

Recall "Formula" Sequences

We think of a formula sequence as a list of numbers, whose typical element is $a(i)$. Given nothing but the value of I , we have a formula that lets us write the value of $a(i)$ immediately.

Examples:

Powers of 2: $a(i) = 2^i$

Triangular: $a(i) = i(i+1)/2$

Interest: $a(i) = \text{original} \cdot (1 + \text{rate})^i$

Factorial: $a(i) = 1 \cdot 2 \cdot 3 \cdot \dots \cdot i$

Sequences as Stepping Stones

Sometimes, however, a formula is not the best way to describe a sequence.

Instead, we are given a rule that tells us how to determine the next element $a(i)$ based on one or more previous values.

This is sometimes called a recurrence formula, or recursion, but it's more memorable to think of it in terms of **stepping stones**.

(A similar idea in mathematical induction.)

Math Moment: Factorials

Let's recall the factorial function, written

$$n! = 1 * 2 * 3 * \dots * n$$

Interesting facts:

It shows up in Taylor series, as in:

$$e(x) = 1 + x + x^2/2! + x^3/3! + \dots$$

It counts permutations, orderings of n objects;

It counts subsets size k of a n -set:

$$n\text{-choose-}k = n! / (k! * (n-k)!)$$

MATLAB command: `value = factorial (n);`

In MATLAB, $21!$ is the last value to be computed exactly.

It's related to the Gamma function.

The Factorial Example

We know the factorial can be described by a formula:

$$a(i) = i! = 1 * 2 * \dots * i$$

Instead, we could give the first value:

$$a(0) = 1$$

and then say how to get the next value:

$$a(i) = a(i-1) * i$$

This "stepping stone" description requires us to start at the first value and take a sequence of steps to reach the desired value.

Same Sequence, Different Rule

By **formula**, we compute $5!$ this way:

$$a(5) = 5! = 5 * 4 * 3 * 2 * 1 = 120.$$

By **stepping stone**, we compute:

$$a(0) = 1$$

$$a(1) = a(0) * 1 = 1$$

$$a(2) = a(1) * 2 = 2$$

$$a(3) = a(2) * 3 = 6$$

$$a(4) = a(3) * 4 = 24$$

$$a(5) = a(4) * 5 = 120$$

Stepping Stone Advantages?

The stepping stone rule seems to require more multiplication to get $5!$, but in this case, it's exactly the same work.

And if we actually wanted to compute a list $a(0)$ through $a(5)$, the stepping stone method uses a total of 5 multiplications, while formulas would use 15.

Stepping stone seems to require setting up all those intermediate variables, but we'll fix that.

How NOT to Compute 5! in MATLAB

```
a0 = 1;  
a1 = a0 * 1;  
a2 = a1 * 2;  
a3 = a2 * 3;  
a4 = a3 * 4;  
a5 = a4 * 5;  
fprintf ( ' 5! = %d\n', a5 );
```

Too much work defining variables. And if we pick a different factorial to compute, we have to write a new program.

Since we are repeating commands, we want to use a FOR or WHILE statement, wrapped around just a few commands.

A Strategy

To compute $5!$ requires 5 steps.

On step I , the new value is computed by multiplying the old value by I .

We can use a FOR loop, $I = 1 : 5$ or, in general, $I = 1 : N$.

Depending on what we need to remember, we can use ONE variable or TWO.

Two Variable Stepping Stone Program

Strategy: A is the "new" value, and $AOLD$ is the previous one. On each step, we copy A into $AOLD$ and compute the next one.

```
a = 1;
```

```
for i = 1 : n
```

```
    aold = a;
```

```
    a = aold * i;
```

```
end
```

One Variable Stepping Stone Program

Strategy: Just update A immediately.

```
a = 1;  
for i = 1 : n  
    a = a * i;  
end
```

Notice that we lose the ability to measure how big our step was. If the sequence is suppose to converge, we would want to compare successive elements, and this would not be easy in the One Variable program.

Bring First Value Inside Loop

It might be better to rewrite the loop so that the starting value is also set inside the loop. It's a little cleaner. It corresponds better to how the rule is defined. And, if we want to print each value, we only need one print statement, not two (because the first value is no longer outside the loop).

Revised One Variable Stepping Stone

The loop limits are 0:n, which explains the range of the sequence.

Inside the loop, we have a nice presentation of the stepping stone rule.

```
for i = 0 : n

    if ( i == 0 )
        a = 1
    else
        a = a * i;
    end

    fprintf ( ' %d', a );
end
fprintf ( '\n' );
```

Insight Through

factorial_formula.m

```
% factorial_formula.m
% Show how to compute a sequence of factorials, using a formula.
%
nmax = input ( 'Enter NMAX, highest factorial to compute: ' );

for n = 0 : nmax

    fact = 1;
    for i = 1 : n
        fact = fact * i;
    end

    fprintf ( ' %d', fact );

end

fprintf ( '\n' );
```

Insight Through

factorial_stepping.m

```
% factorial_stepping.m
% Show how to compute a sequence of factorials, using stepping stones.
%
nmax = input ( 'Enter NMAX, highest factorial to compute: ' );

for n = 0 : nmax

    if ( n == 0 )    ← First step is special
        fact = 1;
    else            ← After that, use the "stepping stone" rule
        fact = fact * n;
    end

    fprintf ( ' %d', fact );

end

fprintf ( '\n' );
```

Insight Through

factorial_stepping2.m

```
% Show how to compute a sequence of factorials, using stepping stones,  
% and keeping the previous value.  
%  
nmax = input ( 'Enter NMAX, highest factorial to compute: ' );  
  
for n = 0 : nmax  
  
    if ( n == 0 )  
        factold = 0;           ← Just make up a "previous" value on first step;  
        fact = 1;  
    else  
        factold = fact;       ← Save the previous value, perhaps for comparison.  
        fact = factold * n;  
    end  
  
    fprintf ( ' %d', fact );  
  
end  
  
fprintf ( '\n' );
```

Insight Through

Stepping Stones in Computing

Stepping Stone sequences are common in computing.

In most examples we will see, to compute $a(n)$ only requires knowing one old value, $a(n-1)$, although there is at least one famous exception we will talk about soon.

Describing a sequence using stepping stones takes us near the computational solution of **differential equations**, simulating the behavior of a quantity like temperature by computing a sequence of values.

Stepping Stone Examples

FACT: $a(0)$ is 1, and $a(i) = a(i-1) * i$;

TRI: $a(0)$ is 0, and $a(i) = a(i-1) + i$;

EVEN: $a(0)$ is 0, and $a(i) = a(i-1) + 2$;

POW2: $a(0)$ is 1, and $a(i) = a(i-1) * 2$;

8CHOOSEI: $a(0)$ is 1, and $a(i) = a(i-1) * (8-i) / i$;

NCHOOSE2: $a(0)$ is 1, and $a(i) = a(i-1) + i$;

INT: $a(0)$ is PRINCIPAL and $a(i) = (1 + RATE) * a(i-1)$;

Can we work out these sequences?

New Rule, Old Tasks

If we are using a stepping stone rule for a sequence, let's make sure we can still:

- * Print the first N entries;
- * Plot the first N entries;
- * Compute entries UNTIL some condition;
- * Compute the MAX;

Print a Stepping Stone Sequence

interest_sequence.m

```
rate = input ( 'Enter the interest rate: ' );
year_end = input ( 'Enter the final year: ' );

for year = 2017 : year_end
    if ( year == 2017 )
        amount = 1000.0;           ← initialize inside loop
    else
        amount = amount * ( 1.0 + rate ); ← stepping stone rule
    end
    fprintf ( '%d %.0f\n', year, amount );
end
```

Plot a Stepping Stone Sequence

interest_plot.m

```
rate = input ( 'Enter the interest rate: ' );
year_end = input ( 'Enter the final year: ' );

amount_list = []; ← initialize empty list
for year = 2017 : year_end
    if ( year == 2017 )
        amount = 1000.0; ← initialize inside loop
    else
        amount = amount * ( 1.0 + rate ); ← stepping stone rule
    end
    fprintf ( '%d %.0f\n', year, amount );
    amount_list = [ amount_list, amount ]; ← update list
end

plot ( 2017:year_end, amount_list ); ← list of years, list of amounts
```

Run Stepping Stone Sequence UNTIL interest_until.m

```
rate = input ( 'Enter the interest rate: ' );
year = 2017;;

while ( true )

    if ( year = 2017 )
        amount = 1000.0;
    else
        amount = amount * ( 1.0 + rate ); ← stepping stone rule
    end

    if ( 5000 <= amount )
        break
    end

    year = year + 1;

end

fprintf ( 'In %d, $%0.f exceeds $5000\n', year, amount );
```

Insight Through

Computing the MAX

So far, almost all our sequences consist of numbers that grow larger and larger, so that the last number computed is the largest.

This is not always the case; a sequence might oscillate up and down, and in that case, we might be interested in knowing the maximum value observed.

Remember, although we have computed N entries, we only see one or two values at a time, so when we are done, it's too late to say "Just compute the max now!"

The MAX of a Stepping Stone Sequence

```
for k = 0 : n
```

```
    if ( k == 0 )
```

```
        value = initial value;
```

```
        value_max = value;
```

```
    else
```

```
        value = STEPPING STONE FORMULA
```

```
        value_max = max ( value_max, value );
```

```
    end
```

```
end
```

```
fprintf ( 'Maximum value observed = %d\n', value_max );
```

Insight Through

Maximum of Stepping Stone Sequence

nchoosek_max.m

```
n = input ( 'Enter the value of N: ' );

for k = 0 : n

    if ( k == 0 )
        value = 1;
        value_max = value;
    else
        value = value * ( n + 1 - k ) / k;
        value_max = max ( value_max, value );
    end

end

fprintf ( 'Maximum value observed = %d\n', value_max );
```

Mathematics Moment: N-Choose-K

N-Choose-K is how mathematicians describe the number of ways of choosing K distinct objects from a set of N.

MATLAB includes the command:

```
value = nchoosek ( n, k );
```

N-Choose-K is only nonzero for $0 \leq K \leq N$, so this set of numbers forms a sort of **finite sequence**.

Choosing 2 things from a set of 10 is the same as choosing 8 things, because you are really splitting the set into two pieces: chosen and unchosen.

This means the N-Choose-K sequence is **symmetric**.

By convention, N-Choose-K is 1 if K is 0.

N-Choose-K

The formula for N-Choose-K:

$$N\text{-Choose-K} = n! / (k! * (n-k)!)$$

assuming $0 \leq K \leq N$, 0 otherwise.

$$4\text{Choose}2 = 4! / (2! * 2!) = 24 / (2 * 2) = 6$$

$$5\text{Choose}3 = 5! / (3! * 2!) = 120 / (6 * 2) = 10$$

$$6\text{Choose}0 = 6! / (0! * 6!) = 720 / (1 * 720) = 1$$

N-Choose-K for N = 4

Computing the values of N-Choose-K when N = 4:

$$\text{N-Choose-0} = 1$$

$$\text{N-Choose-1} = 4$$

$$\text{N-Choose-2} = 6$$

$$\text{N-Choose-3} = 4$$

$$\text{N-Choose-4} = 1$$

Or, as a single line: 1, 4, 6, 4, 1

Formula for N-Choose-K

Compare 1, 4, 6, 4, 1 to the formula:

$$\text{N-Choose-K} = n! / (k! * (n-k)!)$$

for $N = 4$, $K = 0, 1, 2, 3, 4$.

Can we verify these values?

When $N = 5$, can we compute the values of N-Choose-K for $K = 0, 1, 2, 3, 4, 5$?

nchoosek_formula.m

```
n = input ( 'Enter the value of N: ' );  
k = input ( 'Enter the value of K: ' );  
  
value = factorial ( n ) / ...  
    ( factorial ( k ) * factorial ( n - k ) );  
  
fprintf ( ' %d-Choose-%d = %d\n' , n, k,  
    value );
```


Stepping Stone for N-Choose-K

Can we find a stepping stone rule for N-Choose-K that explains 1,4,6,4,1?

K=0: the value is 1

K=1: multiply by N (remember, $N = 4$).

K=2: how does 4 become 6? Try
multiplication: $4 * 3/2 = 6$.

K=3: 6 becomes 4? Maybe $6 * 2/3 = 4$;

K=4: 4 becomes 1? Multiply by $\frac{1}{4}$.

Looking for Pattern

1

$$4 = 1 * 4 = 1 * 4/1$$

$$6 = 4 * 3/2 = 6 * 3/2$$

$$4 = 6 * 2/3 = 6 * 2/3$$

$$1 = 4 * 1/4 = 4 * 1/4$$

Looks like we multiply by $4/1, 3/2, 2/3$
 $1/4$

Does the Pattern Really Work?

Check it out on next line, N = 5:

1, 5, 10, 10, 5, 1

N=5, K=0 1 (starting value)

N=5, K=1: $1 * 5/1 = 5$

N=5, K=2: $5 * _ / _ = 10$

N=5, K=3: $_ * _ / _ = 10$

N=5, K=4: $_ * _ / _ = 5$

N=5, K=5: $_ * _ / _ = 1$

N-Choose-K (stepping stone)

To compute N-Choose-K:

Set value = 1;

Multiply value by $n+1-1 / 1$;

Multiply value by $n+1-2 / 2$;

...

Multiply value by $n+1-k / k$;

nchoosek_stepping.m

```
n = input ( 'Enter the value of N: ' );
k = input ( 'Enter the value of K: ' );

for i = 0 : k

    if ( i == 0 )
        value = 1;
    else
        value = value * ( n + 1 - i ) / i;
    end

end

fprintf ( ' %d-Choose-%d = %d\n', n, k, value );
```

Explaining the Steppingstone Pattern

Look at row 6:

1, 6, 15, 20, 15, 6, 1

15 is row $N=6$, column $K=2$, and $15 = 6! / (2! * 4!)$

20 is row $N=6$, column $K=3$ and $20 = 6! / (3! * 3!)$

Do you see how the formula for 15 becomes the formula for 20 if we...divide by 3 and multiply by 4?

Our Stepping Stone Rule

So an N-Choose-K value can be computed by a complicated formula, but if we know $A(K-1)$, the value for $K-1$, we can compute $A(K)$ in a very simple way:

$$A(K) = A(K-1) * (N+1-K)/K.$$

That's the stepping stone rule we've been using, but now we know **why it works**.

What Happens as We Vary N?

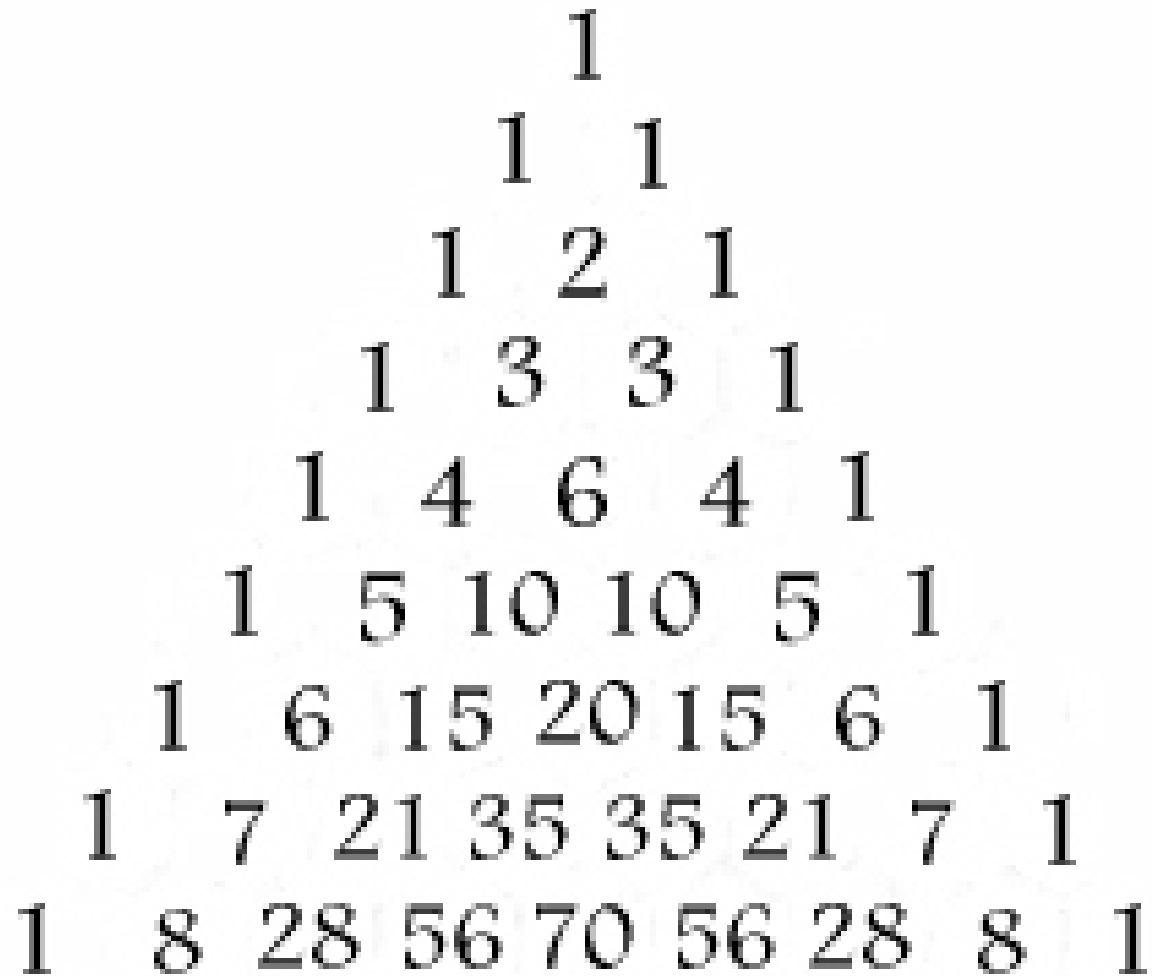
For any N, we can ask for the value of N-Choose-K for $K = 0, 1, 2, \dots, N$, that is, for N+1 values.

For each N, we can write these N+1 numbers as a row.

Because the rows get longer with N, they form a triangle, known as Pascal's triangle.

Thus, our 1,4,6,4,1 values are row 4 of this triangle.

Pascal's Triangle



Pascal's Triangle is a triangular array of binomial coefficients. Each number is the sum of the two numbers directly above it. The triangle is symmetric about its center.

				1																
				1		1														
				1		2		1												
				1		3		3		1										
				1		4		6		4		1								
				1		5		10		10		5		1						
				1		6		15		20		15		6		1				
				1		7		21		35		35		21		7		1		
				1		8		28		56		70		56		28		8		1

Ways to Compute Pascal's Triangle

We can compute N-Choose-K, in row N, column K by:

a) $c_{nk} = nchoosek (n, k);$

b) $c_{nk} = n! / (k! * (n-k)!)$

$= factorial(n) / (factorial(k) * factorial(n-k));$

c) stepping stone formula, starting at 1 and multiplying by $n/1, (n-1)/2, \dots, (n-k+1)/k.$

d) add the two values in the previous row, just above the slot for N-Choose-K!

Printing Pascal's Triangle

Rows and Columns start with index 0. In terms of N-Choose-K, N counts the row, and K the column:

K=0 K=1 K=2 K=3 K=4

N = 0: 0,0

N = 1: 1,0 1,1

N = 2: 2,0 2,1 2,2

N = 3: 3,0 3,1 3,2 3,3

N = 4: 4,0 4,1 4,2 4,3 4,4

To print this triangle, we must handle rows $N = 0 : NMAX$, and in row N , columns $K = 0 : N$. We'll need a pair of FOR loops, the outer one picks the row N , then the inner one runs through the columns indexed by K .

pascal_triangle.m

```
nmax = input ( 'Enter maximum row to print: ' );
```

```
for n = 0 : nmax
```

```
    for k = 0 : n
```

```
        if ( k == 0 )
```

```
            value = 1;
```

```
        else
```

```
            value = value * ( n + 1 - k ) / k;
```

```
        end
```

```
        fprintf ( ' %3d', value );
```

```
    end
```

```
    fprintf ( '\n' );
```

```
end
```

Mathematical Fact

The sum of the entries in row N of Pascal's triangle is 2^N .

Given a sequence that we are computing, how would we go about computing the sum of all the entries we have seen so far?

Let's figure this out by verifying the Math fact!

pascal_rowsum.m

```
n = input ( 'Enter row to check: ' );  
  
sum = 0;  
  
for k = 0 : n  
  
    if ( k == 0 )  
        value = 1;  
    else  
        value = value * ( n + 1 - k ) / k;  
    end  
  
    sum = sum + value;  
  
end  
  
fprintf ( 'Row %d sums to %d, and we expected %d\n', n, sum, 2^n );
```

Insight Through

Pascal and Coin Flipping

Row N of Pascal's triangle can be interpreted as the number of ways of getting K heads in N tosses of a fair coin.

Row 7 tells us there are 7 ways of getting 1 head in 7 tosses, but 35 ways of getting 3 heads. That means 3 heads are 5 times as likely as 1 head, when carrying out 7 tosses.

MATLAB has a `bar()` command that can display the resulting values the values in a row of the matrix, giving us a sense for how these values vary.

pascal_rowbar.m

```
n = input ( 'Enter row to plot: ' );
```

```
value_list = [];
```

```
for k = 0 : n
```

```
    if ( k == 0 )
```

```
        value = 1;
```

```
    else
```

```
        value = value * ( n + 1 - k ) / k;
```

```
    end
```

```
    value_list = [ value_list, value ];
```

```
end
```

```
bar ( 0:n, value_list )
```

← we can add grid, title, xlabel, ylabel...

Insight Through

Probabilities

Row N of Pascal's triangle gives us the number of ways a fair coin, tossed N times, will result in K heads.

The total number of ways is 2^N .

So if we are interested in reporting the probability of K heads, we just divide all the entries in row N by 2^N .

This time, the Y axis of the bar plot will represent actual probability.

Let's Consider Exercise Questions

- 1) What is the next row of the triangle?
- 2) How does this triangle tell you how to write out $(x+y)^4$?
- 3) In 4 coin tosses, how many ways can I get 2 Heads, 2 Tails?
- 4) What is the probability of 2 Heads, and 2 Tails, in 4 coin tosses?
- 5) How many ways can I choose 5 things from a set of 7?
- 6) Is there a formula for the number in row N, column K?
- 7) Is there a way to write out the N-th row of this triangle, without any other information?
- 8) If a 256 steel balls drop down through a pyramid of nails involving 9 rows, what pattern are they likely to form?

To Learn More:

Martin Gardner, "The multiple charms of Pascal's triangle", *Scientific American*, December, 1966.

"The Galton board",
<https://www.youtube.com/watch?v=6YDHBFVIVIs>

Homework #3

Due by Midnight, Friday, September 22

hw012: rewrite a positive number in scientific notation, using two `WHILE` statements.

hw027: estimate an infinite alternating decreasing infinite series, using a `WHILE` statement.

hw028: pay your grocery bill at an automatic checkout, using `WHILE`.

Homework #4

Due by Midnight, Friday, October 6

hw029: approximate the golden ratio by summing part of an infinite series.

hw030: how long must a penny fall before it reaches the center of the earth?

hw032: when will a typical child weight 1000 pounds, according to Theron?

Exam

We will have an in-class exam on
Thursday, September 28th.

The exam counts as 15% of your grade.

It will be a written exam, involving short
answers or short MATLAB scripts.

A practice exam will be available by
Tuesday; I may post a copy in Canvas
before then, in the "files" subdirectory
named 09_28.