2.1-2.3 Applications: [Condition Number and Hill Cipher]
by Dr. Sarah

Condition Number (2.3)

Maple and other computer algebra programs can compute a ***condition number*** for a square matrix**.**

The condition number of the identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (1)$$

```
> with(LinearAlgebra): with(plots): ConditionNumber([[1,0],[0,1]]);
```

$$1 \quad (2)$$

Control click on the output and convert to scientific notation.

The condition number of a matrix that is not invertible (a singular matrix) is undefined:

```
> ConditionNumber([[1,2,3],[4,5,6],[7,8,9]]);
Error, (in LinearAlgebra:-MatrixInverse) singular matrix
```

The condition number of $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9.000000100 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \frac{90000001}{10000000} \end{bmatrix}$ is large!

```
> ConditionNumber([[1,2,3],[4,5,6],[7,8,90000001/10000000]]);
```

$$\frac{11519999807999999}{6000000} \quad (3)$$

```
> evalf(%);
```

$$1.919999968\ 10^{9} \quad (4)$$

The condition number measures the asymptotically worst case of how much the function can change in proportion to small changes in the argument. As a general rule of thumb, condition number is $\sim 10^{9}$ (ie the order k is 9) then we may lose up to 9 digits of accuracy.

Let's reexamine a matrix we had problems with in the past. Here I took the first three rows from the Coffee Mixing linear combination analysis we previously did, to make it a square matrix:

```
> CoffeeMixing := Matrix([[.3, .4, 36], [.2, .3, 26], [.2, .2, 20]]);
```

$$CoffeeMixing := \begin{bmatrix} 0.3 & 0.4 & 36 \\ 0.2 & 0.3 & 26 \\ 0.2 & 0.2 & 20 \end{bmatrix} \quad (5)$$

```
> ReducedRowEchelonForm(CoffeeMixing);
```

(6)

$$\begin{bmatrix} 1. & 0. & 0. \\ -0. & 1. & -0. \\ -0. & -0. & 1. \end{bmatrix} \quad (6)$$

This gives us full (column and row) pivots, suggesting that the Coffee Mixing matrix is invertible, or equivalently, that the columns of the matrix are linearly independent (or any of the other conditions from Theorem 8 since the matrix is square).

However, this is false:
col3= 40*col1 + 60*col 2
shows that the columns are NOT linearly independent.

We previously resolved this with fractions:

```
> FractionsCoffeeMixing := Matrix([[3/10, 4*(1/10), 36], [2*(1/10),
  3/10, 26], [2*(1/10), 2*(1/10), 20]]);
```

$$FractionsCoffeeMixing := \begin{bmatrix} \frac{3}{10} & \frac{2}{5} & 36 \\ \frac{1}{5} & \frac{3}{10} & 26 \\ \frac{1}{5} & \frac{1}{5} & 20 \end{bmatrix} \quad (7)$$

```
> ReducedRowEchelonForm(FractionsCoffeeMixing);
```

$$\begin{bmatrix} 1 & 0 & 40 \\ 0 & 1 & 60 \\ 0 & 0 & 0 \end{bmatrix} \quad (8)$$

To understand this at a deeper level, notice that:

```
> MatrixInverse(FractionsCoffeeMixing);
Error, (in MatrixInverse) singular matrix
> ConditionNumber(FractionsCoffeeMixing);
Error, (in LinearAlgebra:-MatrixInverse) singular matrix
> MatrixInverse(CoffeeMixing);
```

$$\begin{bmatrix} -9.00719925474105\ 10^{16} & 9.00719925474105\ 10^{16} & 4.50359962737052\ 10^{16} \\ -1.35107988821116\ 10^{17} & 1.35107988821116\ 10^{17} & 6.75539944105579\ 10^{16} \\ 2.25179981368526\ 10^{15} & -2.25179981368527\ 10^{15} & -1.12589990684263\ 10^{15} \end{bmatrix} \quad (9)$$

```
> ConditionNumber(CoffeeMixing);
```

$$1.239615798\ 10^{19} \quad (10)$$

A program may not be able to numerically distinguish between a singular matrix (a matrix that is not invertible) and one with a large condition number. Row reduction or the inverse method may produce errors as a result of roundoff error.

This is an issue with the matrix, not with Maple.

The condition number measures the asymptotically worst case of how much the function can change in proportion to small changes in the argument. As a general rule of thumb,
condition number is $\sim 10^{19}$ (ie the order k is 19) then you may lose up to 19 digits of accuracy.

This is obviously not very good for fields like engineering where all measurements, constants and inputs are approximate. You'll be investigating this in the Problem Set.

However, the condition number can also be used for good news---
to tell you how many digits you need to use (r) to (usually) be accurate to r-k digits.

So if we want to be accurate to 1 decimal in computations like
$\vec{x}$=MatrixInverse(CoffeeMixing).$\vec{b}$
to solve
CoffeeMixing $\vec{x}=\vec{b}$
use an accuracy of at least 20 decimals in CoffeeMixing and $\vec{b}$, because 20-19=1.

There is more information about the condition number at the end of Chapter 6 and in Chapter 7, if you are interested in investigating this for the final project.

Hill Cipher

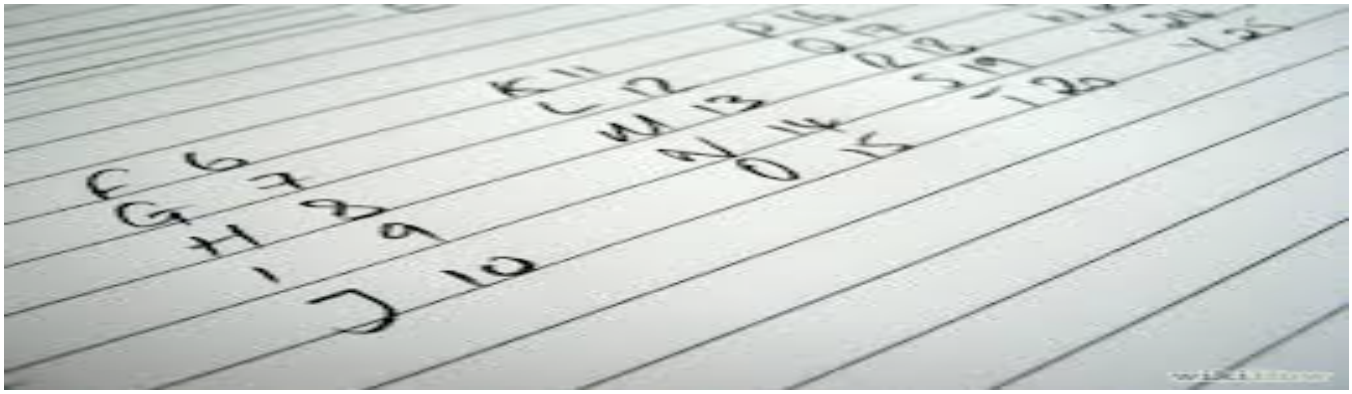
The Hill Cipher is a standard application of 2.1-2.3.

```
> Coding:=Matrix([[1,1,1,1,1,1],[1,2,3,4,5,6],[1,3,6,10,15,21],[1,4,
  10,20,35,56],[1,5,15,35,70,126],[1,6,21,56,126,252]]);
```

$$Coding := \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 6 & 10 & 15 & 21 \\ 1 & 4 & 10 & 20 & 35 & 56 \\ 1 & 5 & 15 & 35 & 70 & 126 \\ 1 & 6 & 21 & 56 & 126 & 252 \end{bmatrix} \quad \textbf{(11)}$$

This is a 6x6 Tartaglia matrix, named for Italian algebraist Niccolò Fontana Tartaglia (1500–77) who is often credited with the general formula for solving cubic polynomials. The rows are diagonals of Pascal's triangle.

Turn a message we want to code into numerical vectors of the correct size, using a standard correspondance:
_ --> 0
A -->1
...
Z-->26



CAREFUL: The message goes in as column vectors.

To code the message, we apply A=**Coding matrix** to each vector:
A.[uncoded vector] = [coded vector]. Because **[Ab_1 Ab_2 ... Ab_n]=AB**, we can put the entire message into a matrix (the message goes in as column vectors - not rows), and compute AB, reading the code as the resulting column vectors:

```
> v1:=Vector([1,2,3,4,5,6]); v2:=Vector([7,8,9,10,11,12]);
```

$$v1 := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

$$v2 := \begin{bmatrix} 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} \quad \textbf{(12)}$$

```
> Message:=Matrix([v1,v2]);
```

$$Message := \begin{bmatrix} 1 & 7 \\ 2 & 8 \\ 3 & 9 \\ 4 & 10 \\ 5 & 11 \\ 6 & 12 \end{bmatrix} \quad \textbf{(13)}$$

```
> CodedMessage:=Coding.Message;
```

$$CodedMessage := \begin{bmatrix} 21 & 57 \\ 91 & 217 \\ 266 & 602 \\ 630 & 1386 \\ 1302 & 2814 \\ 2442 & 5214 \end{bmatrix} \quad (14)$$

So the coded message is the string of numbers read down each column.

How do we decode the message once we receive it?

We apply all the algebra from 2.1 and 2.2:
Coding.[uncoded message] = [coded message]

So apply the inverse to both sides:
MatrixInverseCoding(Coding.[uncoded message]) = MatrixInverseCoding.[coded message]
Now associativity: (MatrixInverseCoding.Coding).[uncoded message] = MatrixInverseCoding.[coded message]
Next cancel A with its inverse: I.[uncoded message] = MatrixInverseCoding.[coded message]
Finally, reduce I: [uncoded message] = MatrixInverseCoding.[coded message]

```
> MatrixInverse(Coding).CodedMessage;
```

$$\begin{bmatrix} 1 & 7 \\ 2 & 8 \\ 3 & 9 \\ 4 & 10 \\ 5 & 11 \\ 6 & 12 \end{bmatrix} \quad (15)$$

In the Problem Set you will be solving for a 2x2 decoding matrix (if it exists) to try and break a code.
You'll know the coded message AND (only) part of the decoded message.

```
> DecodingMatrix:=Matrix([[a, b],[c,d]]);
```

$$DecodingMatrix := \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad (16)$$

We'll apply the 2x2 *DecodingMatrix* to *CodedVector_1* and set that equal to the corresponding vector for the decoded part of the message (*DecodedVector_1*).
where the decoded vector arises from intercepting part of a message. Vectors go in as numerical columns.
So *DecodingMatrix.CodedVector_1 = DecodedVector_1*
and
DecodingMatrix.CodedVector_2 = DecodedVector_2

This now looks a lot like 2.1 practice and clicker questions.
We have 4 equations and 4 unknowns so you can solve that any way you like, like reducing a 4x5

augmented matrix.

The basic idea is that 2 vector pairs is enough to either solve for *a*, *b, c* and *d* in the 2x2 *DecodingMatrix* or to see that the system is inconsistent (and to decode the message if it is consistent).