Implementing Gaussian Elimination

We implement **Gaussian Elimination** in three stages of increasing sophistication,
following the development in **Olver and Shakiban**, "**Applied Linear Algebra**," Chapter 1.

The simplest case occurs when we always find the pivot positions to be occupied by non-zero elements. Following Olver and Shakiban, we call this the "regular case."
The next simplest case occurs when there will at least be a suitable nonzero pivot available somewhere, but we may have to swap rows in order to move it into position.
This occurs when the matrix is nonsingular.
In the third and final case, we institute a pivoting policy to improve the numerical performance of the algorithm.

NOTE: In this preliminary version, the algorithms are implemented for square matrices only

Gaussian Elimination - Regular Case

The "Regular Case" refers to a square matrix which can be row reduced to an upper triangular matrix by the following algorithm.
The critical point is that there must be a non-zero element in each pivot position for the algorithm to be successful.
We do not resort to swapping rows. A more general case is handled later.
See **Olver and Shakiban**, "**Applied Linear Algebra,**" Section 1.3.

■ gaussianEliminationRegularCase

```
gaussianEliminationRegularCase::Usage =
  "gaussianEliminationRegularCase[a_?squareMatrixQ] returns
    an upper triangular matrix which is row equivalent to the
    regular square matrix a. For this purpose, the matrix a is
    regular if there is always a nonzero element in each pivot
    position. Thus, it is never necesary to swap rows while
    executing the Gaussian Elimination algorithm on such a matrix.";
```

```
gaussianEliminationRegularCase[a_?squareMatrixQ] :=
 Module[{n = Dimensions[a][[1]], b = a},
  Do[
   If[a[[j, j]] == 0,
    Print["Error: gaussianEliminationRegularCase: A is not regular."];
    Abort[],
    Do[
     c := b[[i, j]] / b[[j, j]];
     b[[i]] = b[[i]] - c b[[j]],
     {i, j + 1, n}]],
   {j, 1, n}];
  b]
```

■ squareMatrixQ

```
squareMatrixQ::Usage = "squareMatrixQ[a_?MatrixQ] returns True
    if a is a square matrix, and otherwise returns False.";
```

```
squareMatrixQ[a_?MatrixQ] :=
 Module[{m = Dimensions[a][[1]], n = Dimensions[a][[2]]},
  If[m == n, True, False]]
```

■ Testing.

```
a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

gaussianEliminationRegularCase[a];
% // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{pmatrix}$$

```
a = {{0, 2, 3}, {4, 5, 6}, {7, 8, 9}};

gaussianEliminationRegularCase[a]
```

```
Error: gaussianEliminationRegularCase: A is not regular.
```

```
$Aborted
```

Gaussian Elimination - Nonsingular Case

The following procedure will row reduce a nonsingular square matrix to an upper triangular matrix.
The naive pivoting policy is merely to move the nearest nonzero candidate into the pivot position.
See **Olver and Shakiban**, "**Applied Linear Algebra,**" Section 1.4.

■ gaussianEliminationNonsingularCase

```
gaussianEliminationNonsingularCase::Usage =
  "gaussianEliminationNonsingularCase[a_?squareMatrixQ]
    returns an upper triangular matrix which is row
    equivalent to the nonsingular square matrix a.";
```

```
gaussianEliminationNonsingularCase[a_?squareMatrixQ] :=
 Module[{n = Dimensions[a][[1]], b = a, k, temp, c},
  Do[
   If[allZeroQ[Drop[b[[j]], j - 1]],
    Print["Error: gaussianEliminationNonsingularCase: A is singular."];
    Abort[],
    If[b[[j, j]] == 0,
     k = indexOfFirstNonzeroEntry[Drop[Transpose[b][[j]], j]];
     temp = b[[j]];
     b[[j]] = b[[j + k]];
     b[[j + k]] = temp];
    Do[
     c := b[[i, j]] / b[[j, j]];
     b[[i]] = b[[i]] - c b[[j]],
     {i, j + 1, n}]],
   {j, 1, n}];
  b]
```

■ allZeroQ

```
allZeroQ::Usage = "allZeroQ[row] returns True
    if row is a list of zeros, and False otherwise.";
```

```
allZeroQ[row_] :=
 If[Length[row] == 1,
  First[row] == 0,
  First[row] == 0 && allZeroQ[Rest[row]]]
```

■ indexOfFirstNonzeroEntry

```
indexOfFirstNonzeroEntry::Usage =
  "indexOfFirstNonzeroEntry[ls] returns the
    index of the first nonzero entry in the list ls.";
```

```
indexOfFirstNonzeroEntry[ls_] :=
  If[ls[[1]] ≠ 0,
   1,
   1 + indexOfFirstNonzeroEntry[Rest[ls]]];
```

■ Testing.

```
a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 2}};

gaussianEliminationNonsingularCase[a];
% // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -7 \end{pmatrix}$$

```
a = {{0, 2, 3}, {0, 0, 6}, {3, 8, 2}};

gaussianEliminationNonsingularCase[a];
% // MatrixForm
```

$$\begin{pmatrix} 3 & 8 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & 6 \end{pmatrix}$$

```
a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

gaussianEliminationNonsingularCase[a]
```

```
Error: gaussianEliminationNonsingularCase: A is singular.
```

```
$Aborted
```

Gaussian Elimination - with Partial Pivoting

The following procedure will row reduce a nonsingular square matrix to an upper triangular matrix.
The **Partial Pivoting** version of Gaussian Elimination uses a more sophisticated pivoting policy to reduce accummulating roundoff errors which may lead to inaccurate solutions. The idea is to always move the largest (in absolute value) available element into the pivot position. Here, the "largest available element" means the largest element in the present column which is in or below the present pivot position. A yet more wide-ranging pivoting policy could even use column operations (which effectively only permute the ordering of the variables) to bring in even larger pivots, and further reduce roundoff errors. Such a policy is called **Full Pivoting**, but we do not implement it here. A more sophisticated version of partial or full pivoting would use a list of pointers to rows and pointers to columns and simply **interchange pointers** rather than interchanging whole rows or columns in computer memory.
See **Olver and Shakiban**, "**Applied Linear Algebra,**" Section 1.7.

■ gaussianEliminationPartialPivoting

```
gaussianEliminationPartialPivoting::Usage =
  "gaussianEliminationPartialPivoting[a_?squareMatrixQ] uses
    partial pivoting to return an upper triangular matrix which
    is row equivalent to the nonsingular square matrix a.";
```

```
gaussianEliminationPartialPivoting[a_?squareMatrixQ] :=
 Module[{n = Dimensions[a][[1]], b = a, k, temp, c},
  Do[
   If[allZeroQ[Drop[b[[j]], j - 1]],
    Print["Error: gaussianEliminationPartialPivoting: A is singular."];
    Abort[],
    k = indexOfLargestAvailableEntry[Drop[Transpose[b][[j]], j - 1]];
    If[k > 0,
     temp = b[[j]];
     b[[j]] = b[[j + k]];
     b[[j + k]] = temp];
    Do[
     c := b[[i, j]] / b[[j, j]];
     b[[i]] = b[[i]] - c b[[j]],
     {i, j + 1, n}]],
   {j, 1, n}];
  b]
```

■ allZeroQ

```
allZeroQ::Usage = "allZeroQ[row] returns True
    if row is a list of zeros, and False otherwise.";
```

```
allZeroQ[row_] :=
 If[Length[row] == 1,
  First[row] == 0,
  First[row] == 0 && allZeroQ[Rest[row]]]
```

■ indexOfLargestAvailableEntry

```
indexOfLargestAvailableEntry::Usage =
  "indexOfLargestAvailableEntry[ls] returns the index of the
    element in the list ls having the largest absolute value.";
```

```
indexOfLargestAvailableEntry[ls_] :=
 Module[{max = Max[Abs[ls]]},
  If[Abs[ls[[1]]] == max,
   0,
   1 + indexOfLargestAvailableEntry[Rest[ls]]]]
```

■ Testing.

```
a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 2}};

gaussianEliminationPartialPivoting[a];
% // MatrixForm
```

$$\begin{pmatrix} 7 & 8 & 2 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & \frac{7}{2} \end{pmatrix}$$

```
a = {{0, 2, 3}, {0, 0, 6}, {3, 8, 2}};

gaussianEliminationPartialPivoting[a];
% // MatrixForm
```

$$\begin{pmatrix} 3 & 8 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & 6 \end{pmatrix}$$

```
a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

gaussianEliminationPartialPivoting[a]
```

```
Error: gaussianEliminationPartialPivoting: A is singular.
```

```
$Aborted
```