Pensieve header: Non-Commutative Gaussian Elimination.

The Commutative Case

```
vs = Table[(i + j)^4, {i, 12}, {j, 8}];
vs // Column
```

```
{16, 81, 256, 625, 1296, 2401, 4096, 6561}
{81, 256, 625, 1296, 2401, 4096, 6561, 10000}
{256, 625, 1296, 2401, 4096, 6561, 10000, 14641}
{625, 1296, 2401, 4096, 6561, 10000, 14641, 20736}
{1296, 2401, 4096, 6561, 10000, 14641, 20736, 28561}
{2401, 4096, 6561, 10000, 14641, 20736, 28561, 38416}
{4096, 6561, 10000, 14641, 20736, 28561, 38416, 50625}
{6561, 10000, 14641, 20736, 28561, 38416, 50625, 65536}
{10000, 14641, 20736, 28561, 38416, 50625, 65536, 83521}
{14641, 20736, 28561, 38416, 50625, 65536, 83521, 104976}
{20736, 28561, 38416, 50625, 65536, 83521, 104976, 130321}
{28561, 38416, 50625, 65536, 83521, 104976, 130321, 160000}
```

```
MatrixRank[vs]
```

```
5
```

```
n = 8;
Do[t@i = Null, {i, n}]
```

```
Feed[v_List] /; Total[Abs[v]] == 0 := Null;
```

```
?...
```

p ... or RepeatedNull[*p*] is a pattern object that represents a sequence of zero or more expressions, each matching *p*. ≫

```
?MatchQ
```

MatchQ[*expr*, *form*] returns True if the pattern *form* matches *expr*, and returns False otherwise.
MatchQ[*form*] represents an operator form of MatchQ that can be applied to an expression. ≫

```
MatchQ[{1, 2}, {0..}]
```

```
False
```

```
MatchQ[{0, 0, 0, 0}, {0..}]
```

```
True
```

```
x == y
```

```
x == y
```

```
x == 1
```

```
x == 1
```

```
Solve[x^2 - 1 == 0, x]
```

```
{{x → -1}, {x → 1}}
```

```
Solve[x^2 - 1 === 0, x]
```

```
{}
```

```
Solve[True, x]
```

```
{{}}
```

```
Solve[x^2 - 1 == 0, {x, y}]
```

```
Solve::svars : Equations may not give solutions for all "solve" variables. >>
```

```
{{x → -1}, {x → 1}}
```

```
x^2 - 1 == 0 /. Solve[x^2 - 1 == 0, x][[1]]
```

```
True
```

```
True /. Solve[True, x][[1]]
```

```
True
```

```
Feed[{0..}] := Null;
Feed[v_List] := Module[{i},
  i = 1; While[v[[i]] == 0, ++i];
  If[t[i] === Null,
   t[i] = v / v[[i]],
   Feed[v - t[i] v[[i]]]
  ]
 ]
```

```
Feed /@ vs
```

$$\left\{\left\{1, \frac{81}{16}, 16, \frac{625}{16}, 81, \frac{2401}{16}, 256, \frac{6561}{16}\right\},\right.$$
$$\left\{0, 1, \frac{10\,736}{2465}, \frac{29\,889}{2465}, \frac{13\,312}{493}, \frac{1517}{29}, \frac{45\,360}{493}, \frac{371\,441}{2465}\right\},$$
$$\left\{0, 0, 1, \frac{82\,749}{18\,866}, \frac{113\,880}{9433}, \frac{249\,985}{9433}, \frac{477\,705}{9433}, \frac{1\,660\,071}{18\,866}\right\},$$
$$\left\{0, 0, 0, 1, \frac{13\,996}{3013}, \frac{39\,850}{3013}, \frac{89\,420}{3013}, \frac{173\,495}{3013}\right\},$$
$$\left.\{0, 0, 0, 0, 1, 5, 15, 35\}, \text{Null}, \text{Null}, \text{Null}, \text{Null}, \text{Null}, \text{Null}, \text{Null}\right\}$$

```
? t
```

```
Global`t
```

$$t[1] = \left\{1, \frac{81}{16}, 16, \frac{625}{16}, 81, \frac{2401}{16}, 256, \frac{6561}{16}\right\}$$

$$t[2] = \left\{0, 1, \frac{10736}{2465}, \frac{29889}{2465}, \frac{13312}{493}, \frac{1517}{29}, \frac{45360}{493}, \frac{371441}{2465}\right\}$$

$$t[3] = \left\{0, 0, 1, \frac{82749}{18866}, \frac{113880}{9433}, \frac{249985}{9433}, \frac{477705}{9433}, \frac{1660071}{18866}\right\}$$

$$t[4] = \left\{0, 0, 0, 1, \frac{13996}{3013}, \frac{39850}{3013}, \frac{89420}{3013}, \frac{173495}{3013}\right\}$$

$$t[5] = \{0, 0, 0, 0, 1, 5, 15, 35\}$$

$$t[6] = \text{Null}$$

$$t[7] = \text{Null}$$

$$t[8] = \text{Null}$$

? Count

Count[*list*, *pattern*] gives the number of elements in *list* that match *pattern*.
Count[*expr*, *pattern*, *levelspec*] gives the total number
of subexpressions matching *pattern* that appear at the levels in *expr* specified by *levelspec*.
Count[*pattern*] represents an operator form of Count that can be applied to an expression. ≫

```
Count[t /@ Range[n], _List]
```

5

? Position

Position[*expr*, *pattern*] gives a list of the positions at which objects matching *pattern* appear in *expr*.
Position[*expr*, *pattern*, *levelspec*] finds only objects that appear on levels specified by *levelspec*.
Position[*expr*, *pattern*, *levelspec*, *n*] gives the positions of the first *n* objects found.
Position[*pattern*] represents an operator form of Position that can be applied to an expression. ≫

```
Position[{1, 0, 1, 0, 0, 3, 5}, 0]
```

{{2}, {4}, {5}}

```
Position[{1, 0, 1, 0, 0, 3, 5}, _, {1}]
```

{{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}}

? Except

Except[*c*] is a pattern object which represents any expression except one that matches *c*.
Except[*c*, *p*] represents any expression that matches *p* but not *c*. ≫

```
Position[{1, 0, 1, 0, 0, 3, 5}, Except[0]]
```

{{0}, {1}, {3}, {6}, {7}, {}}

```
? FirstPosition
```

FirstPosition[*expr*, *pattern*] gives the position of the first element in *expr* that matches *pattern*, or Missing["NotFound"] if no such element is found.
FirstPosition[*expr*, *pattern*, *default*] gives *default* if no element matching *pattern* is found.
FirstPosition[*expr*, *pattern*, *default*, *levelspec*] finds only objects that appear on levels specified by *levelspec*.
FirstPosition[*pattern*] represents an operator form of FirstPosition that can be applied to an expression. ≫

```
FirstPosition[{1, 0, 1, 0, 0, 3, 5}, Except[0], Heads → False]
```

```
{1}
```

```
t[19] = Nothing
```

```
Nothing
```

```
t[19]
```

```
Nothing
```

```
Table[t[i], {i, 17, 22}]
```

```
{t[17], t[18], t[20], t[21], t[22]}
```

On to Rubik's Cube

```
? Cycles
```

Cycles[{cyc_1, cyc_2, …}] represents a permutation with disjoint cycles cyc_i. ≫

```
n = 54;
g1 = Cycles[
    {{1, 18, 45, 28}, {2, 27, 44, 19}, {3, 36, 43, 10}, {46, 52, 54, 48}, {47, 49, 53, 51}}];
g2 = Cycles[{{7, 16, 39, 30}, {8, 25, 38, 21}, {9, 34, 37, 12},
     {13, 15, 33, 31}, {14, 24, 32, 22}}];
g3 = Cycles[{{28, 31, 34, 48}, {29, 32, 35, 47}, {30, 33, 36, 46},
     {37, 39, 45, 43}, {38, 42, 44, 40}}];
g4 = Cycles[{{1, 3, 9, 7}, {2, 6, 8, 4}, {10, 54, 16, 13}, {11, 53, 17, 14}, {12, 52, 18, 15}}];
g5 = Cycles[
    {{1, 13, 37, 46}, {4, 22, 40, 49}, {7, 31, 43, 52}, {10, 12, 30, 28}, {11, 21, 29, 19}}];
g6 = Cycles[{{3, 48, 39, 15}, {6, 51, 42, 24}, {9, 54, 45, 33},
     {16, 18, 36, 34}, {17, 27, 35, 25}}];
```

```
? PermutationProduct
```

PermutationProduct[*a*, *b*, *c*] gives the product of permutations *a*, *b*, *c*. ≫

```
a_ ∘ b_ := PermutationProduct[a, b]
```

```
? InversePermutation
```

InversePermutation[*perm*] returns the inverse of permutation *perm*. ≫

? PermutationSupport

PermutationSupport[*perm*] returns the support of the permutation *perm*. ≫

? PermutationReplace

PermutationReplace[*expr*, *perm*] replaces each part in *expr* by its image under the permutation *perm*.

PermutationReplace[*expr*, *gr*] returns the list of images of *expr* under all elements of the permutation group *gr*. ≫