Definitions

```
SetDirectory[NotebookDirectory[]];
```

Eventually, these functions may want to use the Notation` package to actually make the subscripted letters have downvalues. For now, I'm just being careful.

```
Clear@ClearSubscripts
ClearSubscripts[top_] := (
  (Unset[Evaluate[#[[1]]]];
     Evaluate[#[[1]]]) & /@ Select[DownValues[Subscript], #[[1, 1, 1]] == top &]
 )
Clear@unknownArray
unknownArray[top_, dim_ /; NumberQ[dim]] := (
  Table[top_i, {i, dim}]
 )
unknownArray[top_, {dim1_ /; NumberQ[dim1], dim2_ /; NumberQ[dim2]}] := (
  Table[top_{i,j}, {i, dim1}, {j, dim2}]
 )
```

Investment

Pt 1

```
A = {{"Chris", 100, 100, 100}, {"Rebecca", 100, 200, 200}, {"Siddhartan", 50, 50, 200}};
stocks = {1, $100.00, $50.00, $20.00};

A.stocks // MatrixForm
A.stocks /. x_ + y_ :> {x, y} // MatrixForm
```

$$\begin{pmatrix} \text{Chris} + \$17\,000.00 \\ \text{Rebecca} + \$24\,000.00 \\ \text{Siddhartan} + \$11\,500.00 \end{pmatrix}$$

$$\begin{pmatrix} \text{Chris} & \$17\,000.00 \\ \text{Rebecca} & \$24\,000.00 \\ \text{Siddhartan} & \$11\,500.00 \end{pmatrix}$$

Pt 2

```
ClearSubscripts[x];
unknown = unknownArray[x, {3, 3}];
money = {{1500, 1600, 1400}, {2600, 2810, 2550}, {950, 1020, 1000}};
prices = {{100, 110, 100}, {50, 50, 40}, {20, 22, 30}};
eqn = unknown.prices == money;

Solve[eqn, Variables[eq]][[1]]
(unknown /. %) // MatrixForm
```

$$\{x_{1,1} \to 10,\ x_{1,2} \to 10,\ x_{1,3} \to 0,\ x_{2,1} \to 20,\ x_{2,2} \to 10,\ x_{2,3} \to 5,\ x_{3,1} \to 5,\ x_{3,2} \to 5,\ x_{3,3} \to 10\}$$

$$\begin{pmatrix} 10 & 10 & 0 \\ 20 & 10 & 5 \\ 5 & 5 & 10 \end{pmatrix}$$

Physics

Helper function: 2D scalar cross product

```
Clear@cross;
cross[a_, b_] := (Append[a, 0] × Append[b, 0])[[3]]
```

Step 1: calculate the equivalent force (and location) of the lift.

```
Clear@liftforce;
liftforce[x_] := 200 Sqrt[1 - x^2 / 17]
intercept = x /. Solve[{liftforce[x] == 0, x > 0}, x][[1]];
{liftmag, liftpos} =
 Integrate[{1, x} * liftforce[x], {x, 0, intercept}] /. {t_, x_} :> {t, x / t}
```

$$\left\{50\sqrt{17}\,\pi,\ \frac{4\sqrt{17}}{3\pi}\right\}$$

Step 2: Define the forces (and associated radii) for each force and moment.
Force 1 is between the wing and the fuselage
Force 2 is between the strut and wing
Force 3 is lift

```
F1 = unknownArray[f1, 2];
F2 = unknownArray[f2, 2];
F3 = {0, liftmag};
r1 = {0, 0};
r2 = {2, 0};
r3 = {liftpos, 0};
```

Setup motion equations

```
forceeqn = Thread[F1 + F2 + F3 == {0, 0}];
momenteqn = m + cross[r1, F1] + cross[r2, F2] + cross[r3, F3] == 0;
```

Solution strategy 1: use Solve[]

```
Solve[{forceeqn, momenteqn}, Variables@{forceeqn, momenteqn}]
```

$$\left\{\left\{f1_2 \to \frac{m}{2} - \frac{50}{3}\left(-34 + 3\sqrt{17}\,\pi\right),\ f2_1 \to -f1_1,\ f2_2 \to -\frac{1700}{3} - \frac{m}{2}\right\}\right\}$$

Solution strategy 2: use Reduce[]

```
Reduce[{forceeqn, momenteqn}, Variables@{forceeqn, momenteqn}]
```

$$f1_1 == -f2_1 \,\&\&\, f1_2 == -50\sqrt{17}\,\pi - f2_2 \,\&\&\, m == -\frac{3400}{3} - 2\,f2_2$$

Solution strategy 3: use a matrix

```
vars = {f1_1, f2_1, f1_2, f2_2, m};
{b, a} = Normal@CoefficientArrays[Flatten@{forceeqn, momenteqn}, vars];
HoldForm@A == (a // MatrixForm)
HoldForm@b == (b // MatrixForm)
```

$$A == \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{pmatrix}$$

$$b == \begin{pmatrix} 0 \\ 50\sqrt{17}\,\pi \\ \frac{3400}{3} \end{pmatrix}$$

Now, the solution has been reduced to the form [A.vars=b].

```
Solve[a.vars == b, vars]
```

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

$$\left\{\left\{f2_1 \to -f1_1,\ f2_2 \to 50\sqrt{17}\,\pi - f1_2,\ m \to -\frac{100}{3}\left(-34 + 3\sqrt{17}\,\pi\right) + 2\,f1_2\right\}\right\}$$

Electronics

I' ll come back to this after hearing the in class explanation. It is probably useful, so I' m sorry I didn't solve it.

Types of Solutions

Solving with Elimination

This function carries out rotely the elimination algorithm for 2-d equation solvers given. Notice that I use x and y instead of x_1 and x_2, simply for brevity.

```
Clear@elim2
elim2[eq1_, eq2_] := Module[{soleq1, soleq2, xval, yval},
  soleq1 = Solve[eq1, {x}][[1]];
  soleq2 = eq2 /. soleq1;
  yval = y /. Solve[soleq2, y][[1]];
  xval = soleq1[[1, 2]] /. y → yval;
  {xval, yval}
 ]
```

Note that the solution given in the BB is $\begin{pmatrix} 1 \\ 3/2 \end{pmatrix}$, an erroneous typo.

```
elim2[2 x + 3 y == 6, 4 x + 9 y == 15] // MatrixForm
```

$$\begin{pmatrix} \frac{3}{2} \\ 1 \end{pmatrix}$$

This equation has infinitely many solutions:

```
elim2[x + 2 y == 1, 2 x + 4 y == 2] // MatrixForm
```

$$\begin{pmatrix} 1 - 2\,y \\ y \end{pmatrix}$$

The result here states that as long as y==y (trivially true) and x==1-2y, the equations are satisfied.

This equation has no solutions.

```
elim2[x + 2 y == 1, 2 x + 4 y == 1] // MatrixForm
```

```
Part::partw : Part 1 of {} does not exist. >>

ReplaceAll::reps :
  {{}[[1]]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing. >>
```

$$\begin{pmatrix} 1 - 2\,(y /. \{\}[[1]]) \\ y /. \{\}[[1]] \end{pmatrix}$$

As expected, warnings are thrown and a nonsensical result is returned. In particular, the first solution to an equation with no solutions doesn't exist. I'd like the error to be more intuitive, but this works.

Solving for h, k

```
elim2[x + h y == 1, 2 x + 3 y == k] // MatrixForm
```

$$\begin{pmatrix} 1 - \frac{h\,(2-k)}{-3+2\,h} \\ \frac{2-k}{-3+2\,h} \end{pmatrix}$$

Interpreting this result is a little hoakey. In particular, it is clear that *something* interesting happens when $h = \frac{3}{2}$, but it takes some thought to realize that the result is no solution because $\frac{n}{0}$ is undefined for nonzero n. On the other hand, if $h = \frac{3}{2}$ && $k = 2$, then the y value is $\frac{0}{0}$, which allows for infinitely many solutions.

Amazingly, this code was written once, then applied to all of the cases without needing to be tested, altered, or generalized. I think that's interesting.

Three variables

This code is a pretty simple extension of the stuff above, but demonstrates the added complexity from solving higher-order equations using elimination.

```
Clear@elim3
elim3[eq1_, eq2_, eq3_] := Module[{},
  soleq1 = Solve[eq1, x][[1]];
  soleq2 = Solve[eq2 /. soleq1, y][[1]];
  zval = z /. Solve[eq3 /. Join[soleq1, soleq2], z][[1]];
  yval = soleq2[[1, 2]] /. z → zval;
  xval = soleq1[[1, 2]] /. {z → zval, y → yval};
  {xval, yval, zval}
 ]
```

When I fed in the equations in the given order, this code broke. I suspect that this is the primary reason elimination isn't a good way for computers to solve these things.

```
elim3[x + y + z == 6, x - 2 z == 4, y + z == 4]
```

```
{2, 5, -1}
```

Result: 1 solution

```
elim3[x + y + z == -6, 2 x + y - z == 18, y - 2 z == 4]
```

$$\left\{\frac{52}{5}, -\frac{48}{5}, -\frac{34}{5}\right\}$$

Result : 1 solution

```
elim3[2 x + y - z == 10, x + y + z == 6, y + z == 2]
```

```
{4, 2, 0}
```

Result : 1 solution

Ummm. Those results are ***not right***. I think I've hit the limits of reasonable programmability of elimination, and while I could certainly work it out by hand, cursory visual inspection reveals the answers are (One, none, many). I'll come back and fix this if I have time.
Lesson learned: Don't do elimination by computer!

Gauss-Jordan Elimination

I've done the by-hand part of this *far too many times* before for my Linear Algebra class, and that was valuable. Doing it again now wouldn't be, so I'm not going to.

```
aug = {{1, 1, 1, 6}, {0, 1, 1, 2}, {1, 0, -2, 4}};
RowReduce[aug] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Result: Single solution

```
aug = {{1, 1, 1, -6}, {2, 1, -1, 18}, {1, 0, -2, 4}};
RowReduce[aug] // MatrixForm
```

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

Result: No solutions

```
aug = {{1, 1, 1, 6}, {2, 1, -1, 10}, {1, 0, -2, 4}};
RowReduce[aug] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & -2 & 4 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Result: Infinitely many solutions

I realize that this is the fuller form, but don't see much point in stopping before achieving the leading 1's coefficient.

LU Decomposition

The LUDecomposition command returns the results in an annoying flattened format. This fixes that. Warning: for convenience, it drops the permutation matrix, causing potentially incorrect results.

```
Clear[LU]
LU[mat_] := Module[{lu, p, c, l, u},
  {lu, p, c} = LUDecomposition[mat];
  l =
   lu SparseArray[{i_, j_} /; j < i → 1, Dimensions@mat] + IdentityMatrix[Length@mat];
  u = UpperTriangularize[lu];
  {l, u}
 ]
```

```
mat = {{1, 1, 1}, {0, 1, 1}, {1, 0, -2}};
Grid[{{"L", "U"}, MatrixForm /@ LU[mat]}]
```

L	U
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & -2 \end{pmatrix}$

My manual calculation agrees.

This next problem encapsulates both the second and third examples (they're identical).

```
mat = {{1, 1, 1}, {2, 1, -1}, {1, 0, -2}};
Grid[{{"L", "U"}, MatrixForm /@ LU[mat]}]
```

LUDecomposition::sing : Matrix {{1, 1, 1}, {2, 1, −1}, {1, 0, −2}} is singular. >>

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

My manual calculation agrees. The singular nature of the matrix does not prevent successful calculation.

Determinant

1

```
mat = {{1, 1, 1}, {0, 1, 1}, {1, 0, -2}};
Grid[{{"L", "U"}, MatrixForm /@ LU[mat]}]
Print["Manual result: -2"]
Det[mat]
```

L	U
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & -2 \end{pmatrix}$

```
Manual result: -2
```

```
-2
```

2/3

```
mat = {{1, 1, 1}, {2, 1, -1}, {1, 0, -2}};
Grid[{{"L", "U"}, MatrixForm /@ LU[mat]}]
Print["Manual result: 0"]
Det[mat]
```

LUDecomposition::sing : Matrix {{1, 1, 1}, {2, 1, −1}, {1, 0, −2}} is singular. >>

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

```
Manual result: 0
```

```
0
```

Inverse

```
Clear@LUinverse;
LUinverse[l_, u_] := Module[{},
  inv = ConstantArray[0, Dimensions@l];
  Do[
   intermediate = LinearSolve[l, UnitVector[Length@l, i]];
   inv[[All, i]] = LinearSolve[u, intermediate];
   , {i, Length@l}];
  inv
 ]
```

1

```
mat = {{1, 1, 1}, {0, 1, 1}, {1, 0, -2}};
{l, u} = LU[mat];
Grid[{{"L", "U"}, MatrixForm /@ {l, u}}]
LUinverse[l, u] // MatrixForm
Inverse[mat] // MatrixForm
```

L	U
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & -2 \end{pmatrix}$

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

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

2-3

Because this equation has potentially no solutions, an inverse matrix should not be findable. Let's try anyway.

```
mat = {{1, 1, 1}, {2, 1, -1}, {1, 0, -2}};
{l, u} = LU[mat];
Grid[{{"L", "U"}, MatrixForm /@ {l, u}}]
LUinverse[l, u] // MatrixForm
Inverse[mat] // MatrixForm
```

LUDecomposition::sing : Matrix {{1, 1, 1}, {2, 1, −1}, {1, 0, −2}} is singular. »

L			U		
1	0	0	1	1	1
2	1	0	0	−1	−3
1	1	1	0	0	0

LinearSolve::nosol : Linear equation encountered that has no solution. »

LinearSolve::nosol : Linear equation encountered that has no solution. »

LinearSolve::nosol : Linear equation encountered that has no solution. »

General::stop : Further output of LinearSolve::nosol will be suppressed during this calculation. »

```
{LinearSolve[{{1, 1, 1}, {0, -1, -3}, {0, 0, 0}}, {1, -2, 1}] LinearSolve[{{1, 1, 1}, {0,
 LinearSolve[{{1, 1, 1}, {0, -1, -3}, {0, 0, 0}}, {1, -2, 1}] LinearSolve[{{1, 1, 1}, {0,
 LinearSolve[{{1, 1, 1}, {0, -1, -3}, {0, 0, 0}}, {1, -2, 1}] LinearSolve[{{1, 1, 1}, {0,
```

Inverse::sing : Matrix {{1, 1, 1}, {2, 1, −1}, {1, 0, −2}} is singular. »

```
Inverse[{{1, 1, 1}, {2, 1, -1}, {1, 0, -2}}]
```

As expected, neither method works. As always, *Mathematica* is throwing lots of errors, and the results are strictly speaking correct, but not useful.

Building Bridges

Triangle Truss

```
a = {{-Sin[30 Degree]   0   -Sin[60 Degree]  0  0  0},
     {-Cos[30 Degree]   0    Cos[60 Degree]  0  0  0},
     { Cos[30 Degree]   1          0         1  0  0},
     { Sin[30 Degree]   0          0         0  1  0},
     {       0         -1   -Cos[60 Degree]  0  0  0},
     {       0          0    Sin[60 Degree]  0  0  1}};
b = Transpose@{{1000, 0, 0, 0, 0, 0}};
```

10.

```
If[Quiet[Head[Inverse[a]] === Inverse], "Indeterminate", "Determinate"]
```

```
Determinate
```

11.

```
RowReduce[Join[a, b, 2]] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -500 \\ 0 & 1 & 0 & 0 & 0 & 0 & 250\sqrt{3} \\ 0 & 0 & 1 & 0 & 0 & 0 & -500\sqrt{3} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 250 \\ 0 & 0 & 0 & 0 & 0 & 1 & 750 \end{pmatrix}$$

```
Inverse[a].b // MatrixForm
```

$$\begin{pmatrix} -500 \\ 250\sqrt{3} \\ -500\sqrt{3} \\ 0 \\ 250 \\ 750 \end{pmatrix}$$

```
LinearSolve[a, b] // MatrixForm
```

$$\begin{pmatrix} -500 \\ 250\sqrt{3} \\ -500\sqrt{3} \\ 0 \\ 250 \\ 750 \end{pmatrix}$$

12.

Because of the way the FBDs are drawn, F_1 ... F_3 represent the amount of tension in the three beams. By that interpretation, the solution implies beams 1 and 3 are in compression, and beam 2 is in tension. In reality, that doesn't make any sense. It turns out that you switched the definitions of beams 2 and 3, so actually the lower element is in tension, as expected.

13.

Adding the additional constraint will make the *x* vector one element longer, necessitating *A* getting wider by one element. That will make it non-square, and will eliminate the possibility of having a unique solution (one more unknown than equation).

Equation 5 becomes $-F_2 - F_3 \cos(60) + R_{cx} = 0$.

```
a = {{-Sin[30 Degree], 0, -Sin[60 Degree], 0, 0, 0, 0},
     {-Cos[30 Degree], 0, Cos[60 Degree], 0, 0, 0, 0},
     {Cos[30 Degree], 1, 0, 1, 0, 0, 0},
     {Sin[30 Degree], 0, 0, 0, 1, 0, 0},
     {0, -1, -Cos[60 Degree], 0, 0, 0, 1},
     {0, 0, Sin[60 Degree], 0, 0, 1, 0}};
```

```
b = Transpose@{{1000, 0, 0, 0, 0, 0}};
```

```
If[Quiet[Head[Inverse[a]] === Inverse], "Indeterminate", "Determinate"]
```

```
Indeterminate
```

```
RowReduce[Join[a, b, 2]] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -500 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 250\sqrt{3} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -500\sqrt{3} \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 250 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 750 \end{pmatrix}$$

I have little idea what PseudoInverse[] does, but it was recommended. Help?

```
Inverse[a].b // MatrixForm; (* Fails *)
PseudoInverse[a].b // MatrixForm
```

Inverse::matsq : Argument

$$\left\{\left\{-\frac{1}{2}, 0, -\frac{\sqrt{3}}{2}, 0, 0, 0, 0\right\}, \left\{-\frac{\sqrt{3}}{2}, 0, \frac{1}{2}, 0, 0, 0, 0\right\}, \left\{\frac{\sqrt{3}}{2}, 1, 0, 1, 0, 0, 0\right\}, \left\{\frac{1}{2}, 0, 0, 0, 1, 0, 0\right\}, \left\{0, -1, -\frac{1}{2}, 0, 0, 0, 1\right\}, \left\{0, 0, \frac{\sqrt{3}}{2}, 0, 0, 1, 0\right\}\right\}$$ at position 1 is not a non–empty square matrix. ≫

$$\begin{pmatrix} -500 \\ \frac{500}{\sqrt{3}} \\ -500\sqrt{3} \\ \frac{250}{\sqrt{3}} \\ 250 \\ 750 \\ -\frac{250}{\sqrt{3}} \end{pmatrix}$$

```
LinearSolve[a, b] // MatrixForm
```

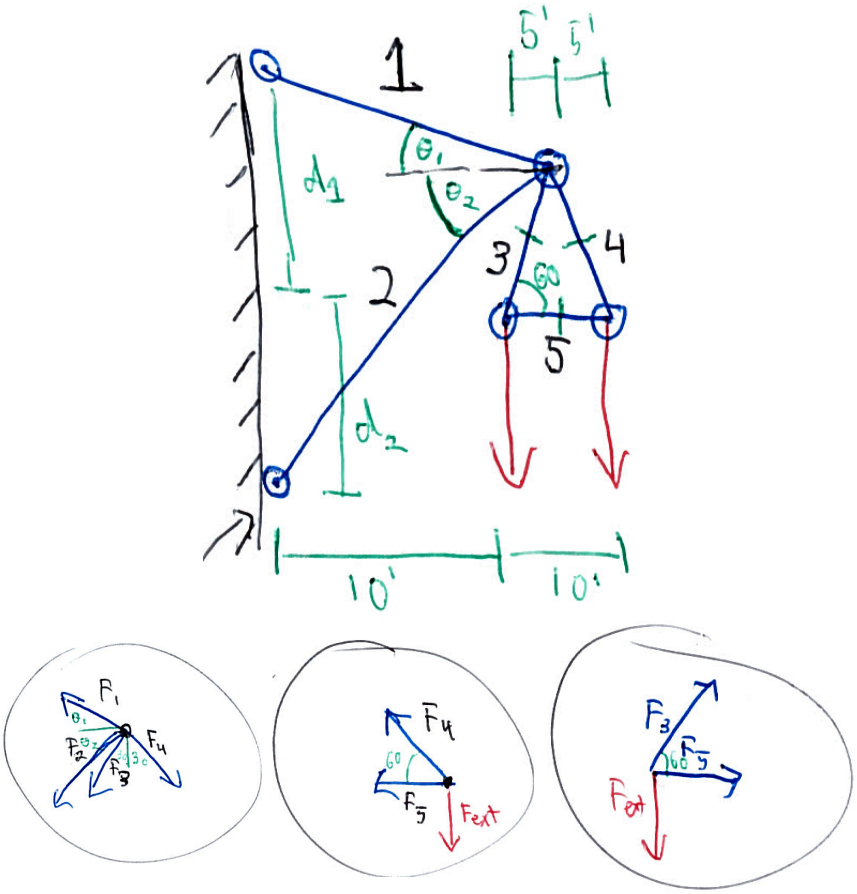
$$\begin{pmatrix} -500 \\ 250\sqrt{3} \\ -500\sqrt{3} \\ 0 \\ 250 \\ 750 \\ 0 \end{pmatrix}$$

This is an indeterminite system, so LinearSolve gives a solution that works. PseudoInverse also gives a solution that works, although Inverse[] fails because the matrix is non-square.

Solving the parallelogram problem was less useful than overachieving on the last problem, so I chose the latter.

Design Problem: Hanging a Sign

The basic design I'm investigating is a rigid triangle mounted to the wall, with a free-hanging equilateral triangle with a mass mounted at each lower vertex. What makes this design *particularly* interesting is that it should not be solvable in any case except the special one where the two external forces are exactly equal. That implies that the matrix *m* must be singular. Despite that, for the special set of cases given, $m.x = b$ should have a singular solution.



Interestingly, modelling the points contacting the wall is unnecessary. Because the reaction forces don't matter to the success conditions, we can just model the three joints on the right.

```
(equations = {
    F4 Sin[30 Degree] - F3 Sin[30 Degree] - F1 Cos[θ1 Degree] - F2 Cos[θ1 Degree] == 0
    (*Joint 1 x*),
    -F4 Cos[30 Degree] - F3 Cos[30 Degree] + F1 Sin[θ1 Degree] - F2 Sin[θ1 Degree] == 0
    (*Joint 1 y*),
    -F4 Cos[60 Degree] - F5 == 0(*Joint 2 x*),
    F4 Sin[60 Degree] - Fext1 == 0(*Joint 2 y*),
    F3 Cos[60 Degree] + F5 == 0(*Joint 3 x*),
    F3 Sin[60 Degree] - Fext2 == 0(*Joint 3 y*)
   }) // MatrixForm
variables = {F1, F2, F3, F4, F5};
```

$$\begin{pmatrix} -\frac{F3}{2}+\frac{F4}{2}-F1\,\text{Cos}[^\circ\theta 1]-F2\,\text{Cos}[^\circ\theta 1]==0 \\ -\frac{\sqrt{3}\,F3}{2}-\frac{\sqrt{3}\,F4}{2}+F1\,\text{Sin}[^\circ\theta 1]-F2\,\text{Sin}[^\circ\theta 1]==0 \\ -\frac{F4}{2}-F5==0 \\ \frac{\sqrt{3}\,F4}{2}-\text{Fext1}==0 \\ \frac{F3}{2}+F5==0 \\ \frac{\sqrt{3}\,F3}{2}-\text{Fext2}==0 \end{pmatrix}$$

```
setup = {θ1 → 30, θ2 → 0, Fext1 → 10 000, Fext2 → 10 000};
```

Method 1: Solve[]

```
Solve[equations /. setup, variables][[1]] // MatrixForm
```

$$\begin{pmatrix} F1 \to 20\,000 \\ F2 \to -20\,000 \\ F3 \to \frac{20\,000}{\sqrt{3}} \\ F4 \to \frac{20\,000}{\sqrt{3}} \\ F5 \to -\frac{10\,000}{\sqrt{3}} \end{pmatrix}$$

Method 2: Matricies

```
(*The inversion of b accomplishes the negation of
 moving the coefficients to the other side of the equation*)
{b, a} = {-1, 1} CoefficientArrays[equations /. setup, variables];
MatrixForm /@ {a, b}
```

$$\left\{\begin{pmatrix} -\frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & -1 \\ 0 & 0 & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 1 \\ 0 & 0 & \frac{\sqrt{3}}{2} & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10\,000 \\ 0 \\ 10\,000 \end{pmatrix}\right\}$$

```
MatrixForm[variables] == MatrixForm[sol = LinearSolve[a, b]]
```

$$\begin{pmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \end{pmatrix} == \begin{pmatrix} 20\,000 \\ -20\,000 \\ \frac{20\,000}{\sqrt{3}} \\ \frac{20\,000}{\sqrt{3}} \\ -\frac{10\,000}{\sqrt{3}} \end{pmatrix}$$

Analysis

Again, all of the values here represent tensions, so these results indicate that bars 1,3, and 4 are in tension, while bars 2 and 5 are under compression. Intuitively, this makes sense.

```
N@sol // MatrixForm
Max[sol] ≤ 45 000
```

$$\begin{pmatrix} 20\,000. \\ -20\,000. \\ 11\,547. \\ 11\,547. \\ -5773.5 \end{pmatrix}$$

```
True
```

Additionally, the results reveal that the maximum tensile load does not exceed the required 45 kips, satisfying the requirements.

Extensions

```
Clear@solve
solve[setup_] :=
 Module[{a, b}, {b, a} = {-1, 1} CoefficientArrays[equations /. setup, variables];
  MatrixForm[variables] == MatrixForm[sol = LinearSolve[a, b]]
  (*Solve[equations/.setup,variables][[1,All,2]]*)
 ]
```

As expected, letting the two external forces be not equal prevents finding a solution.

```
solve[{θ1 → 30, θ2 → 0, Fext1 → 10 000, Fext2 → 9000}]
```

LinearSolve::nosol : Linear equation encountered that has no solution. ≫

$$\begin{pmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \end{pmatrix} == \text{LinearSolve}[\text{SparseArray}[\langle\text{Specified elements: 14, Dimensions: \{6, 5\}}\rangle], \text{SparseArray}[\langle\text{Specified elements: 2, Dimensions: \{6\}}\rangle]]$$

Setting them to floating point (inexact) values also prevents the calculation from proceeding. I do not fully understand why.

```
solve[{θ1 → 30, θ2 → 0, Fext1 → 10 000., Fext2 → 10 000.}]
```

LinearSolve::sqmat :
The matrix SparseArray[≪1≫] is not square. A square matrix is needed to compute a factorization. ≫

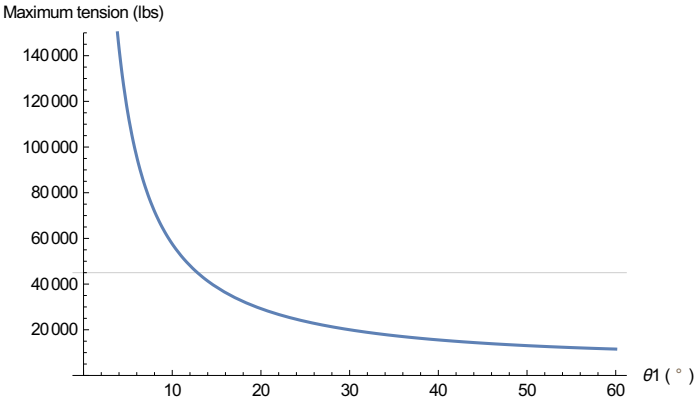
$$\begin{pmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \end{pmatrix} == \text{LinearSolve}[\text{SparseArray}[\langle\text{Specified elements: 14, Dimensions: \{6, 5\}}\rangle], \text{SparseArray}[\langle\text{Specified elements: 2, Dimensions: \{6\}}\rangle]]$$

We can plot the affect of $\theta 1$ on the maximum tension.

A few tricks were brought into play here, namely the conversion of $\theta 1$ to an exact number rather than an approximate value with Round[]. This matrix can *only* be solved in exact terms.

```
maxTension[var_] := (Max[
    solve[{θ1 → Round[var, 10^-1000], θ2 → 0, Fext1 → 10 000, Fext2 → 10 000}][[2, 1]]])
```

```
Plot[maxTension[var], {var, 0, 60}, PlotRange → {0, 150 000},
 GridLines → {{45 000}}, AxesLabel → {"θ1 (°)", "Maximum tension (lbs)"}]
```



Of course, this allows us to Solve for that intersection point...

```
optimum = FindRoot[maxTension[var] == 45 000, {var, 10}]
```

```
{var → 12.8396}
```

This means the entire structure can be optimized to a vertical spread between wall mountpoints of

```
Sin[(var /. optimum) Degree] ft
```

```
0.222222 ft
```

Not bad.

Misc