Matrices, Geometry&*Mathematica*

Authors: Bruce Carpenter, Bill Davis and Jerry Uhl ©2001

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.06 Beyond 3D

BASICS

B.1) Hitting and hanging in higher dimensions

□B.1.a.i) Hitting on 3D and hanging in 2D

Here's a random matrix which hits on 3D points and hangs them in 2D:

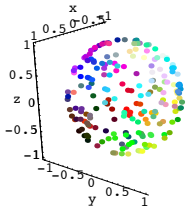
```
hitdim = 3;
hangdim = 2;

Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];
A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];

MatrixForm[A]
```

$$\begin{pmatrix} 1.54136 & 0.732036 & 0.839536 \\ 1.33998 & -1.16661 & 0.175321 \end{pmatrix}$$

```
The number of horizontal rows is the same as hangdim.
The number of vertical columns is the same as hitdim.
```

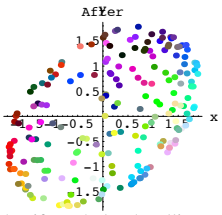
Here are points scattered on the surface of the unit sphere in 3D:

```
hitdim = 3;
points =
  Table[Random[NormalDistribution[0, 1]], {k, 1, 200}, {hitdim}];

spherepoints = Table[points[[k]] / Sqrt[points[[k]].points[[k]]], {k, 1, Length[points]}];

Clear[pointcolor, k];
pointcolor[k_] := RGBColor[0.5 (Sin[N[2 π] spherepoints[[k, 1]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 2]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 3]] + 1)];

spherepointplot = Show[Table[Graphics3D[
     {PointSize[0.03], pointcolor[k], Point[spherepoints[[k]]]}],
    {k, 1, Length[spherepoints]}], Axes → True, Boxed → False,
   AxesLabel → {"x", "y", "z"}, ViewPoint → CMView];
```



Here's what happens to these points after the hit with A:

```
after = Show[Table[Graphics[{PointSize[0.03], pointcolor[k],
      Point[A.spherepoints[[k]]]}], {k, 1, Length[spherepoints]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "After"];
```



Identify and plot the ellipse you see.

□Answer:

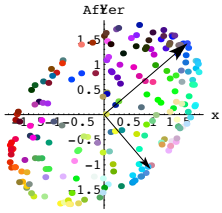
Look at the stretch factors:

```
stretches = SingularValues[A][[2]]
```

{2.1857, 1.42353}

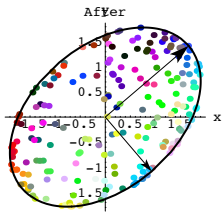
This tells you that a hit with A hangs the results on hangerframe[1] and hangerframe[2].

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

frameup = Show[after, Arrow[stretch[1] hangerframe[1],
    Tail → {0, 0}, VectorColor → Black], Arrow[
    stretch[2] hangerframe[2], Tail → {0, 0}, VectorColor → Black]];
```



Seeing this, you can easily plot the ellipse you want:

```
Clear[ellipseplotter, t];
ellipseplotter[t_] =
 Cos[t] stretch[1] hangerframe[1] + Sin[t] stretch[2] hangerframe[2];

ellipseplot = ParametricPlot[ellipseplotter[t], {t, 0, 2 π},
   PlotStyle → {{Thickness[0.01]}}, DisplayFunction → Identity];

Show[frameup, ellipseplot];
```



There you go.

□B.1.a.ii) Hitting on 4D and hanging in 2D

Here's a random matrix which hits on 4D points and hangs them in 2D:

```
hitdim = 4;
hangdim = 2;

Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];
A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];

MatrixForm[A]
```

$$\begin{pmatrix} 0.326958 & 1.41009 & 0.974031 & -0.58548 \\ 0.611946 & 0.226839 & -0.520292 & 0.827974 \end{pmatrix}$$

```
The number of horizontal rows is the same as hangdim.
The number of vertical columns is the same as hitdim.
```

Here is a table of points scattered on the surface of the unit sphere in 4D shown with the 4D coordinates of the first two:

```
points =
  Table[Random[NormalDistribution[0, 1]], {k, 1, 200}, {hitdim}];

spherepoints = Table[points[[k]] / Sqrt[points[[k]].points[[k]]], {k, 1, Length[points]}];

spherepoints[[1]]
spherepoints[[2]]
```

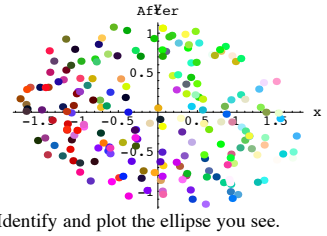
{-0.681813, -0.487302, 0.262304, -0.478398}

{0.260083, 0.586755, -0.129987, 0.755763}

```
Note the four slots.
```

Because these points are in 4D, they cannot be plotted.
But because A hangs in 2D, you can plot what happens when you hit these points with A:

```
Clear[pointcolor, k];
pointcolor[k_] := RGBColor[0.5 (Sin[N[2 π] spherepoints[[k, 1]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 2]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 3]] + 1)];

after = Show[Table[Graphics[{PointSize[0.025], pointcolor[k],
      Point[A.spherepoints[[k]]]}], {k, 1, Length[spherepoints]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "After"];
```



Identify and plot the ellipse you see.

□Answer:

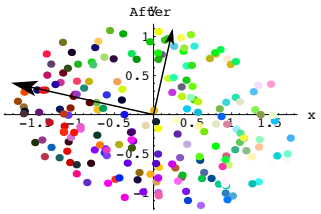
Look at the stretch factors:

```
stretches = SingularValues[A][[2]]
```

{1.86874, 1.12994}

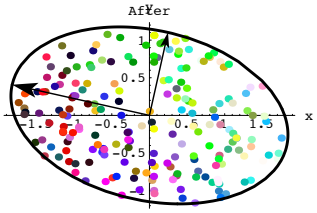
This tells you that a hit with A hangs the results on hangerframe[1] and hangerframe[2].

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

frameup = Show[after, Arrow[stretch[1] hangerframe[1],
    Tail → {0, 0}, VectorColor → Black], Arrow[
    stretch[2] hangerframe[2], Tail → {0, 0}, VectorColor → Black]];
```



Seeing this, you can easily plot the ellipse you want:

```
Clear[ellipseplotter, t];
ellipseplotter[t_] =
 Cos[t] stretch[1] hangerframe[1] + Sin[t] stretch[2] hangerframe[2];

ellipseplot = ParametricPlot[ellipseplotter[t], {t, 0, 2 π},
   PlotStyle → {{Thickness[0.01]}}, DisplayFunction → Identity];

Show[frameup, ellipseplot];
```



No problem-o.

□ B.1.b.i) Hitting on 5D and hanging in 3D

Here's a matrix A which hits on 5D points and hangs them in 3D:

```
A = {{1.257, -1.807, 1.071, 0.985, 1.049},
    {-0.380, -1.946, -1.143, -1.106, 1.562},
    {1.535, 1.813, 1.266, -0.280, -0.695}};

MatrixForm[A]
```

$$\begin{pmatrix} 1.257 & -1.807 & 1.071 & 0.985 & 1.049 \\ -0.38 & -1.946 & -1.143 & -1.106 & 1.562 \\ 1.535 & 1.813 & 1.266 & -0.28 & -0.695 \end{pmatrix}$$

```
The number of horizontal rows is the same as hangdim.
The number of vertical columns is the same as hitdim.
```

Here is a table of points scattered on the surface of the unit sphere in 5D shown with the 5D coordinates of the first two:

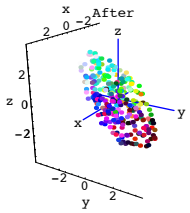
```
hitdim = 5;
hangdim = 3;
points =
  Table[Random[NormalDistribution[0, 1]], {k, 1, 300}, {hitdim}];

spherepoints =
  Table[points[[k]] / Sqrt[points[[k]].points[[k]]], {k, 1, Length[points]}];

spherepoints[[1]]
spherepoints[[2]]
```

```
{-0.303234, -0.231846, 0.601924, -0.643851, 0.27828}
{0.559749, 0.432979, -0.488015, 0.483078, -0.166392}
```

```
Note the five slots.
```

Because these points are in 5D, they can't be plotted, but because A hangs in 3D, you can plot what happens when you hit these points with A:

```
Clear[pointcolor, k];
pointcolor[k_] := RGBColor[0.5 (Sin[N[2 π] spherepoints[[k, 1]]] + 1),
    0.5 (Sin[N[π] spherepoints[[k, 2]]] + 1),
    0.5 (Sin[N[π] spherepoints[[k, 3]]] + 1)];

stretches = SingularValues[A][[2]];

ranger = Max[stretches];
after = Show[Table[Graphics3D[
     {PointSize[0.025], pointcolor[k], Point[A.spherepoints[[k]]]}],
    {k, 1, Length[spherepoints]}], Axes3D[ranger],
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, PlotRange → {{-ranger, ranger},
     {-ranger, ranger}, {-ranger, ranger}}, PlotLabel → "After"];
```



Identify and plot the ellipsoid you see.

□ Answer:

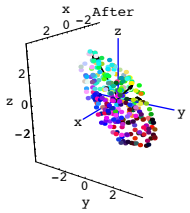
Look at the stretch factors:

```
stretches = SingularValues[A][[2]]
```

```
{3.93723, 2.72089, 1.36606}
```

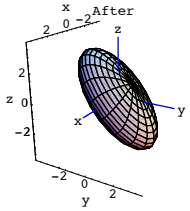
This tells you that a hit with A hangs the results on hangerframe[1], hangerframe[2] and hangerframe[3].

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

frameup = Show[after,
   Table[Arrow[stretch[k] hangerframe[k], Tail → {0, 0, 0},
     VectorColor → Black, ShaftWidth -> 0.01], {k, 1, 3}]];
```



Seeing this, you can easily plot the ellipsoid:

```
Clear[ellipsiodplotter, t];
ellipsoidplotter[s_, t_] =
 Sin[s] Cos[t] stretch[1] hangerframe[1] +
  Sin[s] Sin[t] stretch[2] hangerframe[2] +
  Cos[s] stretch[3] hangerframe[3];

{slow, shigh} = {0, π};

{tlow, thigh} = {0, 2 π};

ellipsoidplot =
 ParametricPlot3D[ellipsoidplotter[s, t], {s, slow, shigh},
  {t, tlow, thigh}, Axes → True, AxesLabel → {"x", "y", "z"},
  Boxed → False, ViewPoint → CMView, PlotLabel → "After", PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  DisplayFunction -> Identity];

Show[ellipsoidplot, Axes3D[ranger],
  DisplayFunction -> $DisplayFunction];
```



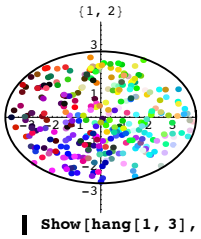

```
Grab all three plots, align and animate.
```

Rock on.

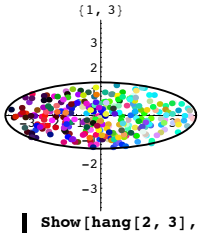
□ B.1.b.ii) Shadows on planes determined by the hanger frames

Stay with the same matrix and the same points as in part i) immediately above and look at these plots:

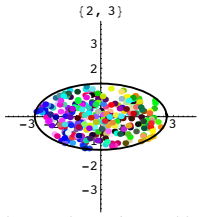
```
Clear[hang, m, n];
hang[m_, n_] :=
  hang[m, n] = Show[Table[Graphics[{PointSize[0.03], pointcolor[k],
       Point[{A.spherepoints[[k]].hangerframe[m],
         A.spherepoints[[k]].hangerframe[n]}]}],
     {k, 1, Length[spherepoints]}], Axes → True, PlotLabel → {m, n},
    PlotRange → {{-ranger, ranger}, {-ranger, ranger}}, Epilog →
     {Thickness[0.01], Circle[{0, 0}, {stretch[m], stretch[n]}]},
    DisplayFunction → Identity];

Show[hang[1, 2], DisplayFunction → $DisplayFunction];
```




```
Show[hang[1, 3], DisplayFunction → $DisplayFunction];
```




```
Show[hang[2, 3], DisplayFunction → $DisplayFunction];
```



What are these plots and how are they related to the ellipsoid and the points plotted in part i)?

□ Answer:

- The component of A.spherepoints[[k]] in the direction of hangerframe[m] is

  (A.spherepoints[[k]].hangerframe[m]) hangerframe[m].

- The component of A.spherepoints[[k]] in the direction of hangerframe[n] is

  (A.spherepoints[[k]].hangerframe[n]) hangerframe[n].

So the projection of A.spherepoints[[k]] onto the plane determined by hangerframe[m] and hangerframe[n] is

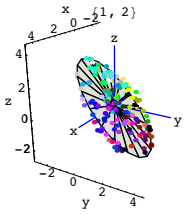
(A.spherepoints[[k]]. hangerframe[m]) hangerframe[m] + (A.spherepoints[[k]]. hangerframe[m]) hangerframe[m].

The plots above use hangerframe[m] and hangerframe[n] as coordinate axes and plot the perpendicular frame coordinates

{(A.spherepoints[[k]]. hangerframe[m]), ((A.spherepoints[[k]]. hangerframe[n]))}

To see what they mean, look at this:

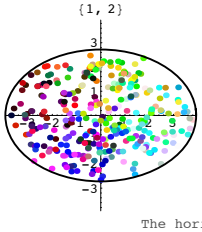
```
Clear[section];
section[m_, n_] :=
   ParametricPlot3D[r (stretch[m] Cos[t] hangerframe[m] +
       stretch[n] Sin[t] hangerframe[n]), {r, 0, 1}, {t, 0, 2 π},
    PlotPoints → {2, Automatic}, PlotLabel → {m, n}, Axes → True,
    AxesLabel → {"x", "y", "z"}, Boxed → False, ViewPoint → CMView,
    PlotRange → All, DisplayFunction → Identity];

Show[section[1, 2], frameup, DisplayFunction → $DisplayFunction];
```



This plotted ellipse is the planar slice of the ellipsoid determined by hangerframe[1] and hangerframe[2].

hang[1,2] plots the shadow (projection) of these points on the ellipse you see above:

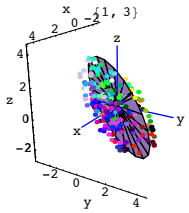
```
Show[hang[1, 2], DisplayFunction → $DisplayFunction];
```



The horizontal axis is defined by hangerframe[1].
The vertical axis is defined by hangerframe[2].

To see what hang[1,3] plots, look at this:

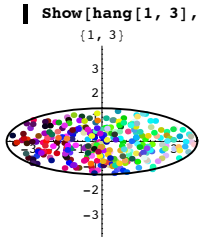
```
Show[section[1, 3], frameup, DisplayFunction → $DisplayFunction];
```



This plotted ellipse is the planar slice of the ellipsoid determined by hangerframe[1] and hangerframe[3].

hang[1,3] plots the shadow (projection) of these points on the ellipse you see above:

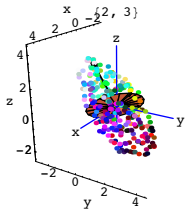
```
Show[hang[1, 3], DisplayFunction → $DisplayFunction];
```



The horizontal axis is defined by hangerframe[1].
The vertical axis is defined by hangerframe[3].

To see what hang[2,3] plots, look at this:

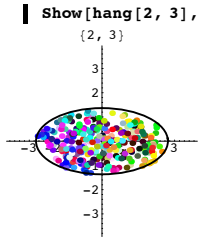
```
Show[section[2, 3], frameup, DisplayFunction → $DisplayFunction];
```



This plotted ellipse is the planar slice of the ellipsoid determined by hangerframe[2] and hangerframe[3].

hang[2,3] plots the shadow (projection) of these points on the ellipse you see above:

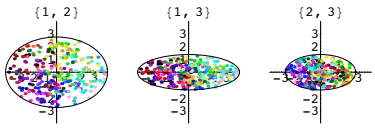
```
Show[hang[2, 3], DisplayFunction → $DisplayFunction];
```



The horizontal axis is defined by hangerframe[2].
The vertical axis is defined by hangerframe[3].

Here they all are:

```
Show[GraphicsArray[{hang[1, 2], hang[1, 3], hang[2, 3]}]];
```



The three main cross sections of the points and the ellipsoid plotted above.

□ B.1.c.i) Hitting on 5D and hanging in 4D

Here's a random matrix which hits on 5D points and hangs them in 4D:

```
hitdim = 5;
hangdim = 4;

Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];

A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];
MatrixForm[A]
```

$$\begin{pmatrix} -0.333208 & 1.25421 & -0.419686 & 0.997183 & -0.146285 \\ -0.806746 & -1.91019 & -1.1629 & -0.450159 & -0.131764 \\ 0.578662 & 1.82958 & 1.80983 & -1.25728 & -1.34829 \\ -0.594026 & -0.528916 & 1.19613 & 0.216982 & -1.25538 \end{pmatrix}$$

The number of horizontal rows is the same as hangdim.
The number of vertical columns is the same as hitdim.

Here is a table of points scattered on the surface of the unit sphere in 5D shown with the 5D coordinates of the first two:

```
points =
  Table[Random[NormalDistribution[0, 1]], {k, 1, 200}, {hitdim}];
```

```
spherepoints =
  Table[points[[k]] / Sqrt[points[[k]].points[[k]]], {k, 1, Length[points]}];

spherepoints[[1]]
spherepoints[[2]]
```

```
{0.501517, -0.124141, 0.697447, -0.452958, 0.203634}
{-0.146282, -0.433501, -0.799486, 0.292952, 0.256282}
```

Note the five slots.

This matrix hits on 5D points and hangs them in 4D.
A sample:

```
A.spherepoints[[1]]
```

```
{-1.09699, -0.801454, 1.62028, 0.248058}
```

Note the four slots.

Because these hit points are in 4D, there is no hope in plotting these points, but you can be sure that they are all packed in an ellipsoid in 4D.
How can you try to look at them?

Answer:

Go to the well again:

```
stretches = SingularValues[A][[2]]
```

```
{3.78675, 2.30756, 1.52352, 0.897354}
```

This tells you that a hit with A hangs points on this 4D perpendicular frame:

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

Table[hangerframe[k], {k, 1, 4}]
```

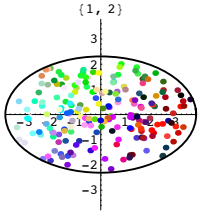
```
{{-0.109718, 0.538865, -0.819624, -0.160635},
 {0.534148, -0.500192, -0.27844, -0.622065},
 {-0.63866, 0.0628122, 0.267645, -0.718703},
 {0.542919, 0.674898, 0.423148, -0.265889}}
```

You cannot plot this frame, but you can do the next best thing.

You can plot the shadow (projection) of the hit points on the two dimensional slicer of 4D defined by hangerframe[m] and hangerframe[n].

Here you go:

```
ranger = Max[stretches];

Clear[pointcolor, k];
pointcolor[k_] := RGBColor[0.5 (Sin[N[2 π] spherepoints[[k, 1]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 2]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 3]] + 1)];

Clear[hang, m, n];
hang[m_, n_] :=
  hang[m, n] = Show[Table[Graphics[{PointSize[0.03], pointcolor[k],
       Point[{A.spherepoints[[k]].hangerframe[m],
         A.spherepoints[[k]].hangerframe[n]}]}],
     {k, 1, Length[spherepoints]}], Axes → True, PlotLabel → {m, n},
    PlotRange → {{-ranger, ranger}, {-ranger, ranger}}, Epilog →
     {Thickness[0.01], Circle[{0, 0}, {stretch[m], stretch[n]}]},
    DisplayFunction → Identity];

Show[hang[1, 2], DisplayFunction → $DisplayFunction];
```



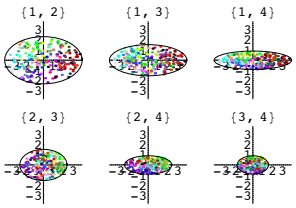
The horizontal axis is defined by hangerframe[1].
The vertical axis is defined by hangerframe[2].

Think of it.

Few have ever been able to peer into 4D with the hit of a key the way you are doing right now.

To see all the 2D cross sections of the 4D ellipsoid you get by hitting the 5D unit sphere with A, just activate:

```
Show[GraphicsArray[{{hang[1, 2], hang[1, 3], hang[1, 4]},
    {hang[2, 3], hang[2, 4], hang[3, 4]}}]];
```



B.1.c.ii) Another high D sample

Does the same thing work for all higher dimensions?

Answer:

Sure does.

Try another:

```
hitdim = 5;
hangdim = 7;


Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];

A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];
MatrixForm[A]
```

$$\begin{pmatrix} -1.61635 & 0.546815 & 1.63467 & 1.60963 & 1.46447 \\ -0.445131 & -0.34351 & -1.82772 & 0.12912 & -0.422711 \\ 0.804958 & -1.96512 & -1.65721 & 0.415925 & -1.14939 \\ -1.94978 & -0.477549 & 1.34967 & 1.42526 & -0.323705 \\ -0.28849 & 1.96791 & -0.526333 & 1.67216 & -0.672139 \\ -0.578905 & -0.161003 & -1.93747 & -0.136614 & 1.86623 \\ -1.81749 & 1.89025 & 1.73427 & 0.288936 & -0.622451 \end{pmatrix}$$

The number of horizontal rows is the same as hangdim.
The number of vertical columns is the same as hitdim.

Here is a table of points scattered on the surface of the unit sphere in 5D shown with the 5D coordinates of the first two:

```
points =
  Table[Random[NormalDistribution[0, 1]], {k, 1, 200}, {hitdim}];

spherepoints =
```

```
  Table[points[[k]] / Sqrt[points[[k]].points[[k]]], {k, 1, Length[points]}];

spherepoints[[1]]
spherepoints[[2]]
```

```
{0.130545, 0.143293, -0.257949, -0.267456, 0.90794}
{0.883079, -0.139094, -0.351855, 0.0285881, -0.276053}
```

Note the five slots.

This matrix hits on 5D points and hangs them in 7D.

A sample:

```
A.spherepoints[[1]]
```

```
{0.344837, -0.0542037, -0.903852,
 -1.34621, -0.677397, 2.13208, -1.05618}
```

Note the seven slots.

Because these hit points are in 7D, there is no hope in plotting these points, but you can be sure that they are all packed in an ellipsoid in 7D.

To get a look at them, go to the well again:

```
stretches = SingularValues[A][[2]]
```

```
{5.45711, 3.03257, 2.80712, 2.69028, 1.50565}
```

There are five nonzero stretch factors here.

This is another way of saying that the rank of A is 5.

This tells you that A hangs its hits on this five dimensional perpendicular frame within 7D.

```
Clear[alignerframe, stretch, hangerframe, k];
alignerframe[k_] := Eigenvectors[Transpose[A].A][[k]];
stretch[k_] := Sqrt[A.alignerframe[k].A.alignerframe[k]];
hangerframe[k_] := A.alignerframe[k] / stretch[k];


rank = 5;
Table[hangerframe[k], {k, 1, rank}]
```

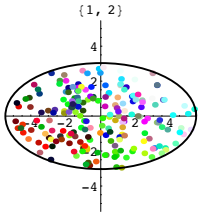
```
{{0.503066, -0.206837, -0.430545, 0.380545,
  0.213486, -0.178259, 0.544617}, {-0.41573, -0.318739,
  0.00846633, -0.117122, -0.181841, -0.808525, 0.158133},
 {-0.217046, 0.382896, 0.379674, 0.137086, 0.755683, -0.1875,
  0.192674}, {-0.240951, -0.0572451, -0.534499, -0.677382,
  0.354399, 0.19372, 0.176079}, {-0.421045, 0.452766,
  -0.106161, 0.187478, -0.42556, 0.219805, 0.584714}}
```

You cannot plot this frame, but you can do the next best thing.

You can plot the shadow (projection) of the hit points on the two dimensional slicer of 7D defined by hangerframe[m] and hangerframe[n].

Here you go:

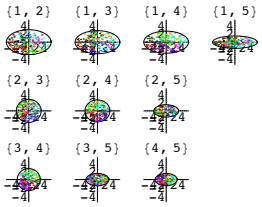
```
ranger = Max[Table[stretch[k], {k, 1, hitdim}]];

Clear[pointcolor, k];
pointcolor[k_] := RGBColor[0.5 (Sin[N[2 π] spherepoints[[k, 1]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 2]] + 1),
   0.5 (Sin[N[π] spherepoints[[k, 3]] + 1)];

Clear[hang, m, n];
hang[m_, n_] :=
  hang[m, n] = Show[Table[Graphics[{PointSize[0.03], pointcolor[k],
       Point[{A.spherepoints[[k]].hangerframe[m],
         A.spherepoints[[k]].hangerframe[n]}]}],
     {k, 1, Length[spherepoints]}], Axes → True, PlotLabel → {m, n},
    PlotRange → {{-ranger, ranger}, {-ranger, ranger}}, Epilog →
     {Thickness[0.01], Circle[{0, 0}, {stretch[m], stretch[n]}]},
    DisplayFunction → Identity];

Show[hang[1, 2], DisplayFunction → $DisplayFunction];
```



The horizontal axis is defined by hangerframe[1].
The vertical axis is defined by hangerframe[2].

To see all of these 5D ellipses within 7D you get by hitting the 5D unit sphere with A, just activate:

```
Show[GraphicsArray[{{hang[1, 2], hang[1, 3], hang[1, 4], hang[1, 5]},
    {hang[2, 3], hang[2, 4], hang[2, 5]},
    {hang[3, 4], hang[3, 5], hang[4, 5]}}]];
```



Life in higher dimensions.

□ B.1.c.iii) Subspaces

Wait a minute. The last matrix hit 5D into 7D but its hangerframe contained only 5 mutually perpendicular vectors:

```
rank = 5;
Table[hangerframe[k], {k, 1, rank}]
```

```
{{0.503066, -0.206837, -0.430545, 0.380545,
  0.213486, -0.178259, 0.544617}, {-0.41573, -0.318739,
  0.00846633, -0.117122, -0.181841, -0.808525, 0.158133},
 {-0.217046, 0.382896, 0.379674, 0.137086, 0.755683, -0.1875,
  0.192674}, {-0.240951, -0.0572451, -0.534499, -0.677382,
  0.354399, 0.19372, 0.176079}, {-0.421045, 0.452766,
  -0.106161, 0.187478, -0.42556, 0.219805, 0.584714}}
```

What happened to the other two dimensions?

□ Answer:

The hit by A on 5D had no chance of filling up all of 7D. When you hit A on all of 5D, you get what folks like to call a five dimensional subspace of 7D.

This subspace consists of all the 7D vectors Y of the form

$$Y = \sum_{k=1}^{\text{rank}} (Y.\text{hangerframe}[k])\ \text{hangerframe}[k]$$

More on this later.

B.2) Rank of a matrix. The advantage you get from full rank.

□ B.2.a.i) Rank is the number of non-zero stretch factors.

What this means

Notice A has been entered through its horizontal rows.

Here's a matrix A which hits on 6D points and hangs them in 5D:

```
A = {{1.257, -1.807, 1.071, 0.985, 1.049, -3.247},
   {-0.380, -1.946, -1.143, -1.106, 1.562, 2.150},
   {1.535, 1.813, 1.266, -0.280, -0.695, -0.945},
   {-1.915, -3.759, -2.409, -0.826, 2.257, 3.095},
   {3.172, 1.952, 3.480, 1.811, -1.208, -6.342}};
MatrixForm[A]
```

$$\begin{pmatrix} 1.257 & -1.807 & 1.071 & 0.985 & 1.049 & -3.247 \\ -0.38 & -1.946 & -1.143 & -1.106 & 1.562 & 2.15 \\ 1.535 & 1.813 & 1.266 & -0.28 & -0.695 & -0.945 \\ -1.915 & -3.759 & -2.409 & -0.826 & 2.257 & 3.095 \\ 3.172 & 1.952 & 3.48 & 1.811 & -1.208 & -6.342 \end{pmatrix}$$

Here are the SVD stretch factors for A:

```
stretches = SingularValues[A][[2]]
```

```
{11.2631, 4.63838, 1.5346}
```

Three positive stretch factors.
This tells you that the rank of A is:

```
rank = Length[stretches]
```

```
3
```

What does this mean?

□ Answer:

Exactly three non-zero stretch factors signals that with A are framed up by a perpendicular frame consisting of 3 5D vectors .

They are:

```
Clear[hangerframe];
{hangerframe[1], hangerframe[2], hangerframe[3]} =
 SingularValues[A][[1]]
```

```
{{-0.221589, 0.298012, -0.219438, 0.51745, -0.739039},
 {0.758425, 0.227791, -0.253714, 0.481505, 0.27692},
 {-0.0262593, -0.695917, -0.715874, 0.0199576, -0.0462168}}
```

Watch it happen for a random hit with A:

```
Clear[x, i];
    x[i_] := Random[Real, {-10, 10}];

randomhit = A.{x[1], x[2], x[3], x[4], x[5], x[6]}
```

```
{-11.1717, -30.6999, 20.6356, -51.3355, 40.1637}
```

To see whether this randomhit is framed by the three hangerframe vectors check whether

$$\text{randomhit} = \sum_{k=1}^{\text{rank}} (\text{randomhit.hangerframe}[k])\ \text{hangerframe}[k]$$

```
Clear[k];
Sum[(randomhit.hangerframe[k]) hangerframe[k], {k, 1, rank}]
```

```
{-11.1717, -30.6999, 20.6356, -51.3355, 40.1637}
```

Rerun both cells a couple of times.

They agree.

And it will happen this way every time because this is what rank means.

□ B.2.a.ii) Another example

Here's a matrix A which hits on 5D points and hangs them in 7D:

```
A = {{1.257, -1.807, 1.071, 0.985, 1.049},
   {-0.380, -1.946, -1.143, -1.106, 1.562},
   {1.535, 1.813, 1.266, -0.280, -0.695},
   {-1.915, -3.759, -2.409, -0.826, 2.257},
   {3.172, 1.952, 3.480, 1.811, -1.208},
   {2.345, 4.169, 3.052, -4.064, 1.094},
   {0.827, -2.217, 0.428, 5.875, -2.302}};
MatrixForm[A]
```

$$\begin{pmatrix} 1.257 & -1.807 & 1.071 & 0.985 & 1.049 \\ -0.38 & -1.946 & -1.143 & -1.106 & 1.562 \\ 1.535 & 1.813 & 1.266 & -0.28 & -0.695 \\ -1.915 & -3.759 & -2.409 & -0.826 & 2.257 \\ 3.172 & 1.952 & 3.48 & 1.811 & -1.208 \\ 2.345 & 4.169 & 3.052 & -4.064 & 1.094 \\ 0.827 & -2.217 & 0.428 & 5.875 & -2.302 \end{pmatrix}$$

Here are the SVD stretch factors for A:

```
stretches = SingularValues[A][[2]]
```

```
{9.82166, 8.43309, 3.53087, 0.569589}
```

Four positive stretch factors. The rank of A is:

```
rank = Length[stretches]
```

```
4
```

What does this mean?

□ Answer:

Exactly four non-zero stretch factors signals that hits with A are framed up by a perpendicular frame consisting of 4 7D vectors .

They are:

```
Clear[hangerframe];
{hangerframe[1], hangerframe[2],
  hangerframe[3], hangerframe[4]} = SingularValues[A][[1]]
```

```
{{0.0657874, 0.182223, -0.267113, 0.449336,
  -0.383549, -0.675794, 0.292245}, {0.125521, -0.233356,
  0.102406, -0.335763, 0.461284, -0.263013, 0.724297},
 {0.722931, 0.368916, -0.019337, 0.388253, 0.334678,
  0.27256, 0.0621183}, {-0.169456, 0.628719, 0.732692,
  -0.103972, -0.0654841, -0.137424, 0.0719402}}
```

Watch it happen for a random hit with A:

```
Clear[x, i];
    x[i_] := Random[Real, {-10, 10}];

randomhit = A.{x[1], x[2], x[3], x[4], x[5]}
```

```
{2.40155, 21.9019, -1.7566, 23.6585, -21.257, 14.3599, -35.6169}
```

To see whether this randomhit is framed by the three hangerframe vectors check whether

$$\text{randomhit} = \sum_{k=1}^{\text{rank}} (\text{randomhit.hangerframe[k]})\ \text{hangerframe[k]}$$

```
Clear[k];
rank
 ∑   (randomhit.hangerframe[k]) hangerframe[k]
k=1
```

```
{2.40155, 21.9019, -1.7566, 23.6585, -21.257, 14.3599, -35.6169}
```

```
Rerun both cells a couple of times.
```

They agree.

And it will happen this way every time because that's what rank means.

□ B.2.b.i) Full rank

Given a matrix A that hits on hitdimD and hangs in hangdimD, what does it mean to say that A is of full rank?
Why do most folks regard full rank as a big bonus?

□ Answer:

Saying that A is of full rank is the same as saying the rank of A is the same as hitdim.

This is the same as saying that none of the SVD stretch factors for A is 0.

This is the same as saying hits with A squash NO non-zero vectors onto {0,0,0,...,0}.

Full rank is a big bonus because if A is of full rank, then you know that corresponding linear systems A.X = Y cannot be under determined (have more than one solution). More on this below.

B.3) Linear algebra:

Given a linear system A.X = Y, put

$$\textbf{Ytest} = \sum_{k=1}^{\text{rank}} (\textbf{Y.hangerframe[k]})\ \textbf{hangerframe[k]}$$

If Ytest ≠ Y, then the linear system A.X = Y has no solution X.

If Ytest = Y, then the linear system A.X = Y has a solution X given by

$$\textbf{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\textbf{Y.hangerframe[k]}}{\textbf{stretch[k]}}\right) \textbf{alignerframe[k].}$$

If A is of full rank, then Xsol is the only solution of A.X = Y (exactly determined).

If A is not of full rank, then Xsol is one of many solutions of A.X = Y (under determined)

□ B.3.a.i) Given a linear system A.X = Y, put

$$\textbf{Ytest} = \sum_{k=1}^{\text{rank}} (\textbf{Y.hangerframe[k]})\ \textbf{hangerframe[k].}$$

If Ytest ≠ Y, then the linear system A.X = Y has no solution for X.

If Ytest = Y, then the linear system A.X = Y has a solution for X

Given a linear system A.X = Y, do an SVD analysis of the coefficient matrix A and put

$$\text{Ytest} = \sum_{k=1}^{\text{rank}} (\text{Y.hangerframe[k]})\ \text{hangerframe[k]}$$

hangerframe[k] is the hangerframe vector corresponding to the kth non-zero stretch factor.

Explain these two facts:
▪ Saying that the linear system A.X = Y has **at least one solution for X** is the same as saying that **Ytest = Y.**

▪ Saying that the linear system A.X = Y has **no solution for X** is the same as saying that **Ytest ≠ Y**

□ Answer

In the problem on rank above, you saw that if Y is a hit with A, then

$$\text{Y} = \sum_{k=1}^{\text{rank}} (\text{Y.hangerframe[k]})\ \text{hangerframe[k]} = \text{Ytest.}$$

So if Y ≠ Ytest, there is no hope of solving A.X = Y for X.

If Y = Ytest, then you will see in part ii) below why

$$\sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}$$

is a guaranteed solution of A.X = Y.

□ B.3.a.ii) Given a linear system A.X = Y, put

$$\textbf{Ytest} = \sum_{k=1}^{\text{rank}} (\textbf{Y.hangerframe[k]})\ \textbf{hangerframe[k].}$$

If Ytest = Y, then a guaranteed solution is

$$\textbf{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\textbf{Y.hangerframe[k]}}{\textbf{stretch[k]}}\right) \textbf{alignerframe[k]}$$

Explain this:
Given a linear system A.X = Y, put

$$\text{Ytest} = \sum_{k=1}^{\text{rank}} (\text{Y.hangerframe[k]})\ \text{hangerframe[k].}$$

If Ytest = Y, then a guaranteed solution is

$$\text{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k].}$$

□ Answer

In this setup, you are given that given that

$$\text{Y} = \text{Ytest} = \sum_{k=1}^{\text{rank}} (\text{Y.hangerframe[k]})\ \text{hangerframe[k].}$$

To do the explanation (proof), just remember that

$$\text{A.alignerframe[k]} = \text{stretch[k]}\ \text{hangerframe[k]},$$

so that

$$\text{hangerframe[k]} = \frac{\text{A.alignerframe[k]}}{\text{stretch[k]}}.$$

This gives

$$\text{Y} = \sum_{k=1}^{\text{rank}} (\text{Y.hangerframe[k]})\ \frac{\text{A.alignerframe[k]}}{\text{stretch[k]}}.$$

Thanks to linearity of matrix hits, this is the same as

$$\text{Y} = \text{A.} \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k].}$$

This signals that

$$\text{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}$$

is a guaranteed solution of the linear system A.X = Y.

□ B.3.a.iii) If Ytest = Y, and A is of full rank, then Xsol is the only solution of A.X = Y (exactly determined).

If Ytest = Y, and A is of not of full rank, then Xsol is not the only solution of A.X = Y (under determined).

Given a linear system A.X = Y with Ytest = Y; so that A.X = Y has at least one solution.
Explain these facts:
▪ If A is **of full rank,** then Xsol is **the only solution** of A.X = Y (exactly determined).
▪ If A is **not of full rank**, then Xsol is **not the only solution** of A.X = Y (under determined).

□ Answer:

▪ Given that A is **of full rank**, you already know that A.Xsol = Y.

The goal is to explain why Xsol is the only solution.

To this end, suppose you've got your hands on another solution Xother. This gives you

$$\text{A.Xother} = \text{Y}$$

and

$$\text{A.Xsol} = \text{Y}.$$

Subtract to get

$$\text{A.Xother} - \text{A.Xsol} = \text{Y} - \text{Y} = \{0, 0, \ldots, 0\}.$$

This gives you

$$\text{A.(Xother} - \text{Xaol)} = \{0, 0, \ldots, 0\}.$$

Because A is of full rank, hits with A squash NO non-zero vectors onto {0,0,...,0}.

This tells you that

$$\text{Xother} - \text{Xsol} = \{0, 0, \ldots, 0\}.$$

And this tells you that

$$\text{Xother} = \text{Xsol}.$$

The upshot:

$$\text{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k].}$$

▪ Given that A is **of not of full rank**, you already know that A.Xsol = Y.

The goal is to explain why Xsol is the not the only solution.

Because A is not of full rank, you are guaranteed that there is an Xzero with

Xzero ≠ {0, 0, . . ., 0}

that is squashed onto {0, 0, . . ., 0} with a hit with A.

In other words,

A.Xzero = {0, 0, . . ., 0}.

This also gives

A.(t Xzero) = t A.Xzero = {0, 0, . . ., 0}

for any number t.

Add this to

A.Xsol = Y

to get

A.Xsol + A.(t Xzero) = Y + {0, 0, . . ., 0} = Y.

for any number t.

This gives

A.(Xsol + t Xzero) = A.Xsol + A.(t Xzero) = Y

for any number t.

As t varies from -infinity to infinity, this gives you infinitely many solutions of

A.X = Y.

□ B.3.b.i) Sample case: Exactly one solution

Here's a matrix which hits on 3D and hangs in 5D:

```
A = {{0.722, -2.855, 2.841},
     {1.315, -0.229, -1.267},
     {2.884, -1.006, 0.789},
     {0.192, 2.081, -2.131},
     {1.929, -1.171, 2.328}};

MatrixForm[A]
```

$$\begin{pmatrix} 0.722 & -2.855 & 2.841 \\ 1.315 & -0.229 & -1.267 \\ 2.884 & -1.006 & 0.789 \\ 0.192 & 2.081 & -2.131 \\ 1.929 & -1.171 & 2.328 \end{pmatrix}$$

This matrix hits on 3D and hangs its hits in 5D.

```
Clear[x, y, z];
A.{x, y, z}
```

```
{0.722 x - 2.855 y + 2.841 z,
 1.315 x - 0.229 y - 1.267 z, 2.884 x - 1.006 y + 0.789 z,
 0.192 x + 2.081 y - 2.131 z, 1.929 x - 1.171 y + 2.328 z}
```

Note the 5 slots.

Use this matrix to make a system of linear equations

A.X = Y:

```
hitdim = 3;
hangdim = 5;

Clear[x, y, k];
X = Table[x[k], {k, 1, hitdim}];
Y = Table[y[k], {k, 1, hangdim}];

linearsystem = ColumnForm[Thread[A.X == Y]]
```

```
0.722 x[1] - 2.855 x[2] + 2.841 x[3] == y[1]
1.315 x[1] - 0.229 x[2] - 1.267 x[3] == y[2]
2.884 x[1] - 1.006 x[2] + 0.789 x[3] == y[3]
0.192 x[1] + 2.081 x[2] - 2.131 x[3] == y[4]
1.929 x[1] - 1.171 x[2] + 2.328 x[3] == y[5]
```

Given a specific

Y = {y[1], y[2], y[3], y[4], y[5]}

saying that the corresponding linear system has a solution is the same as saying that you can find specific numbers x[1], x[2] and x[3] so that when you put

X = {x[1], x[2], x[3]},

then you get

A.X = Y.

Stay with the same linear system as above but go with this specific

Y = {y[1], y[2], y[3], y[4], y[5]}:

```
Y = {24.1787, -6.56968, 11.4069, -16.8513, 20.8961};
hitdim = 3; hangdim = 5;
Clear[x, k];
X = Table[x[k], {k, 1, hitdim}];

linearsystem = ColumnForm[Thread[A.X == Y]]
```

```
0.722 x[1] - 2.855 x[2] + 2.841 x[3] == 24.1787
1.315 x[1] - 0.229 x[2] - 1.267 x[3] == -6.56968
2.884 x[1] - 1.006 x[2] + 0.789 x[3] == 11.4069
0.192 x[1] + 2.081 x[2] - 2.131 x[3] == -16.8513
1.929 x[1] - 1.171 x[2] + 2.328 x[3] == 20.8961
```

If this system has a solution, then come up with one.

Say whether it has additional solutions.

□ Answer:

Do your SVD analysis of the stretch factors of the coefficient matrix A:

```
stretches = SingularValues[A][[2]]
```

```
{6.13718, 3.22015, 1.2119}
```

This determines the rank of A:

```
rank = Length[stretches]
```

```
3
```

The rank of A is the number of hangerframes A uses to hang its hits.
It is also the number of non-zero SVD stretch factors.

Dial up the hangerframe vectors corresponding to the non-zero stretch factors and form Ytest:

```
Clear[hangerframe];
hangerframe[k_] := SingularValues[A][[1]][[k]];
Ytest = Sum[(Y.hangerframe[k]) hangerframe[k], {k, 1, rank}]
```

```
{24.1787, -6.56968, 11.4069, -16.8513, 20.8961}
```

Compare:

```
Y
```

```
{24.1787, -6.56968, 11.4069, -16.8513, 20.8961}
```

Bingo.

Y = Ytest.

This tells you that there is a solution. One guaranteed solution is:

$$\text{Xsol} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}} \right) \text{alignerframe[k].:}$$

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

rank = Length[SingularValues[A][[2]]];

Xsol = Expand[Sum[Y.hangerframe[k] alignerframe[k] / stretch[k], {k, 1, rank}]]
```

```
{1.67029, -0.984556, 7.09674}
```

Check:

```
A.Xsol
Y
```

```
{24.1787, -6.56968, 11.4069, -16.8513, 20.8961}
{24.1787, -6.56968, 11.4069, -16.8513, 20.8961}
```

Good.

To see whether there are more solutions, compare hitdim and the rank:

```
hitdim
```

```
3
```

```
rank
```

```
3
```

Because hitdim = rank, the coefficient matrix A is of full rank.

This tells you the Xsol is the only solution.

□ B.3.b.ii) Sample case: No solutions

Stay with the same linear system as in part i) but go with this specific

Y = {y[1], y[2], y[3], y[4], y[5]}:

```
Y = {-9.62776, 6.26124, 0.436794, 8.02976, -6.23616};
hitdim = 3;
hangdim = 5;

Clear[x, k];
X = Table[x[k], {k, 1, hitdim}];

linearsystem = ColumnForm[Thread[A.X == Y]]
```

```
0.722 x[1] - 2.855 x[2] + 2.841 x[3] == -9.62776
1.315 x[1] - 0.229 x[2] - 1.267 x[3] == 6.26124
2.884 x[1] - 1.006 x[2] + 0.789 x[3] == 0.436794
0.192 x[1] + 2.081 x[2] - 2.131 x[3] == 8.02976
1.929 x[1] - 1.171 x[2] + 2.328 x[3] == -6.23616
```

If this system has a solution, then come up with one.
Say whether it has additional solutions.

□ Answer:

Do your SVD analysis of the stretch factors of the coefficient matrix A:

```
stretches = SingularValues[A][[2]]
```
```
{6.13718, 3.22015, 1.2119}
```

This determines the rank of A:

```
rank = Length[stretches]
```
```
3
```

```
The rank of A is the number of hangerframes A uses to hang its hits.
      It is also the number of non-zero SVD stretch factors.
```

Dial up the hangerframe vectors corresponding to the non-zero stretch factors and form Ytest:

```
Clear[hangerframe];
hangerframe[k_] := SingularValues[A][[1]][[k]];

Ytest = Sum[(Y.hangerframe[k]) hangerframe[k], {k, 1, rank}]
```
```
{-9.59614, 6.20748, 0.490022, 8.06042, -6.29398}
```

Compare:

```
Y
```
```
{-9.62776, 6.26124, 0.436794, 8.02976, -6.23616}
```

Close, but no cigar.

Y is not the same as Ytest.

The call: This linear system has no solution.

□ B.3.b.iii) Sample case: More than one solution

Here's a new matrix A:

```
A = {{0.6146, 0.5210, -0.1295, 1.9475, 1.9705, -0.4572},
     {0.8440, -0.2634, -0.1994, -0.7530, -0.6166, 7.0777},
     {-0.8955, 1.2243, -0.4330, -1.4189, 0.2747, 4.2659},
     {1.2645, -2.7612, 3.9456, 1.4984, 0.1693, -4.5239}};

MatrixForm[A]
```

$$\begin{pmatrix} 0.6146 & 0.521 & -0.1295 & 1.9475 & 1.9705 & -0.4572 \\ 0.844 & -0.2634 & -0.1994 & -0.753 & -0.6166 & 7.0777 \\ -0.8955 & 1.2243 & -0.433 & -1.4189 & 0.2747 & 4.2659 \\ 1.2645 & -2.7612 & 3.9456 & 1.4984 & 0.1693 & -4.5239 \end{pmatrix}$$

```
This matrix hits on 6D and hangs in 4D.
```

Here is a 4D
Y = {y[1], y[2], y[3], y[4]}
together with the corresponding linear system:

```
Y = {10.2452, 2.8353, -8.44979, -5.24377};
hitdim = 6;
hangdim = 4;

Clear[x, k];
X = Table[x[k], {k, 1, hitdim}];

linearsystem = ColumnForm[Thread[A.X == Y]]
```
```
0.6146 x[1] + 0.521 x[2] - 0.1295 x[3] + 1.9475 x[4] + 1.9705 x[5] - 0.4572 x
0.844 x[1] - 0.2634 x[2] - 0.1994 x[3] - 0.753 x[4] - 0.6166 x[5] + 7.0777 x[
-0.8955 x[1] + 1.2243 x[2] - 0.433 x[3] - 1.4189 x[4] + 0.2747 x[5] + 4.2659 x
1.2645 x[1] - 2.7612 x[2] + 3.9456 x[3] + 1.4984 x[4] + 0.1693 x[5] - 4.5239 x
```

If this system has a solution, then come up with one. Say whether it has additional solutions.

□ Answer:

Do your SVD analysis of the stretch factors of the coefficient matrix A:

```
stretches = SingularValues[A][[2]]
```
```
{10.1737, 4.17648, 2.80468, 1.37999}
```

This determines the rank of A:

```
rank = Length[stretches]
```
```
4
```

```
The rank of A is the number of hangerframes A uses to hang its hits.
      It is also the number of non-zero SVD stretch factors.
```

Dial up the hangerframe vectors corresponding to the non-zero stretch factors and form Ytest:

```
Clear[hangerframe];
hangerframe[k_] := SingularValues[A][[1]][[k]];

Ytest = Sum[(Y.hangerframe[k]) hangerframe[k], {k, 1, rank}]
```
```
{10.2452, 2.8353, -8.44979, -5.24377}
```

Compare:

```
Y
```
```
{10.2452, 2.8353, -8.44979, -5.24377}
```

Bingo.

Y = Ytest.

This tells you that there is a solution. One guaranteed solution is:

Xsol = $\sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right)$ alignerframe[k].:

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

rank = Length[SingularValues[A][[2]]];

Xsol = Expand[Sum[Y.hangerframe[k] alignerframe[k] / stretch[k], {k, 1, rank}]]
```
```
{4.00505, -0.951198, -4.75628, 4.5197, -0.53432, 0.187908}
```

Check:

```
A.Xsol
Y
```
```
{10.2452, 2.8353, -8.44979, -5.24377}
{10.2452, 2.8353, -8.44979, -5.24377}
```

Good.

To see whether there are more solutions, compare hitdim and the rank:

```
hitdim
```
```
6
```
```
rank
```
```
4
```

Because hitdim > rank, the coefficient matrix A is of **not** of full rank.

This tells you the Xsol is the not the only solution.

□ B.3.b.iv) Getting all the solutions

Stay with the same linear system as in part i) immediately above.
Come up with all solutions.

□ Answer:

You already know Xsol is a solution. Check:

```
A.Xsol
Y
```
```
{10.2452, 2.8353, -8.44979, -5.24377}
{10.2452, 2.8353, -8.44979, -5.24377}
```

This gives one solution namely:

```
Xsol
```
```
{4.00505, -0.951198, -4.75628, 4.5197, -0.53432, 0.187908}
```

To get the others, look at the rank of A and the hit dim of A

```
rank
hitdim
```
```
4
6
```

The fact that rank < hitdim signals that there are more solutions.

Reason: Hits with A use only the four (= rank of A) aligner frames calculated above and zeroes out 6-4 = 2 others. They are:

```
Clear[zeroedalignerframe];
zeroedalignerframe[k_] := NullSpace[A][[k]];
Table[zeroedalignerframe[k], {k, 1, hitdim - rank}]
```
```
{{-0.758387, -0.392986, -0.0798106, 0.486496, -0.118943, 0.114959},
 {0, -0.630747, -0.338755, -0.422379, 0.555088, -0.0295958}}
```

Check:

```
Table[A.zeroedalignerframe[k], {k, 1, hitdim - rank}]
```
```
{{0, 0, 0, 0}, {0, 0, 0, 0}}
```

All the solutions are parameterized by:

```
Clear[solution, s, t];
solution[s_, t_] =
  Xsol + s zeroedalignerframe[1] + t zeroedalignerframe[2]
```
```
{4.00505 - 0.758387 s, -0.951198 - 0.392986 s - 0.630747 t,
 -4.75628 - 0.0798106 s - 0.338755 t, 4.5197 + 0.486496 s - 0.422379 t,
 -0.53432 - 0.118943 s + 0.555088 t, 0.187908 + 0.114959 s - 0.0295958 t}
```

Try it:

```
Expand[A.solution[s, t]]
Y
```
```
{10.2452, 2.8353, -8.44979, -5.24377}
{10.2452, 2.8353, -8.44979, -5.24377}
```

Perfecto.

B.4) If the linear system A.X = Y has no solutions at all (overdetermined), then

$$\text{Xapprox} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}$$

is the best approximate solution you can get.

In this case, folks call Xapprox by the name "best least squares solution"

□ B.4.a.i) Best approximate solution is $\text{Xapprox} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}$

Here's a linear system:

```
A = {{-0.91, 0.83}, {-0.62, -0.28}, {-0.30, 1.20}};

Y = {-1.05, 0.35, -1.41};

hitdim = 2;
hangdim = 3;

Clear[x, k];
X = Table[x[k], {k, 1, hitdim}];

linearsystem = ColumnForm[Thread[A.X == Y]]
```

$$-0.91\,x[1] + 0.83\,x[2] == -1.05$$
$$-0.62\,x[1] - 0.28\,x[2] == 0.35$$
$$-0.3\,x[1] + 1.2\,x[2] == -1.41$$

In search of a solution X = {x[1], x[2]},you do your SVD analysis of the coefficient matrix A:

```
stretches = SingularValues[A][[2]];
rank = Length[stretches]
```

2

```
The rank of A is the number of hangerframes A uses to hang its hits.
      It is also the number of non-zero SVD stretch factors.
```

Dial up the hangerframe vectors corresponding to the non-zero stretch factors and form Ytest:

```
Clear[hangerframe];
hangerframe[k_] := SingularValues[A][[1]][[k]];
Ytest = Sum[(Y.hangerframe[k]) hangerframe[k], {k, 1, rank}]
```

{-1.01685, 0.316251, -1.4408}

Compare:

```
Y
```

{-1.05, 0.35, -1.41}

Y ≠ Ytest; so there is no **exact** solution.

What do you do to come up with the best **approximate** solution you can get?

□ Answer:

Look at Y and Ytest again:

```
Y
Ytest
```

{-1.05, 0.35, -1.41}

{-1.01685, 0.316251, -1.4408}

Ytest is not all that different from Y and you can find Xapprox with A.Xapprox = Ytest.

You can get it from the same formula use to get solutions. Just put

$$\text{Xapprox} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}.$$

with full assurance that A.Xpprox = Ytest

```
Clear[alignerframe, stretch, hangerframe, k];
hangerframe[k_] := SingularValues[A][[1]][[k]]
stretch[k_] := SingularValues[A][[2]][[k]]
alignerframe[k_] := SingularValues[A][[3]][[k]];

rank = Length[SingularValues[A][[2]]];

Xapprox = Expand[Sum[(Y.hangerframe[k])/stretch[k] alignerframe[k], {k, 1, rank}]]
```

{0.0288925, -1.19345}

And check:

```
A.Xapprox
Y
```

{-1.01685, 0.316251, -1.4408}

{-1.05, 0.35, -1.41}

Xapprox is the best approximate solution you can get.

If you want to see why Xapprox is the best go to the next part.

□ B4.a.ii) Why that worked

Stay with the same setup as in part i) and explain why Xapprox is the best approximate solution of the linear system A.X = Y.is so close to Y.

□ Answer:

All possible hits with A are hung on the frame consisting of:

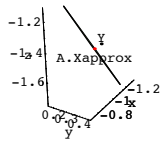
```
Table[hangerframe[k], {k, 1, rank}]
```

{{0.709466, 0.0556394, 0.70254}, {-0.389657, -0.79967, 0.456831}}

This plots out as a plane in 3D.

Here is a plot of that plane in the vicinity of A.Xapprox shown together with a plot of Y:

```
h = 0.2;
k = 0.2;
planeplot = ParametricPlot3D[
   A.(Xapprox + s alignerframe[1] + t alignerframe[2]), {s, -h, h},
   {t, -k, k}, PlotPoints → {2, 2}, DisplayFunction → Identity];

Ypointplot = Graphics3D[{PointSize[0.03], Point[Y]}];
Ylabel = Graphics3D[Text["Y", Y + {h/3, k/3, k/3}]];

AXapproxpointplot =
  Graphics3D[{Red, PointSize[0.03], Point[A.Xapprox]}];

Ylabel = Graphics3D[Text["Y", Y + {0, 0, k/5}]];
AXapproxlabel =
  Graphics3D[Text["A.Xapprox", A.Xapprox - {0, 0, k/3}]];

setup = Show[planeplot, Ypointplot, Ylabel,
   AXapproxpointplot, AXapproxlabel, Axes → True,
   AxesLabel → {"x", "y", "z"}, Boxed → False, ViewPoint → CMView,
   PlotRange → All, DisplayFunction → $DisplayFunction];
```



The vector running from A.Xapprox to Y seems to be perpendicular to the plane on which A hangs its hits. Confirm:

```
(Y - A.Xapprox).hangerframe[1]
```

0

```
(Y - A.Xapprox).hangerframe[2]
```

0

This tells you that Y - A.Xapprox is perpendicular to both the vectors that frame the plane on which A hangs its hits.

This is enough to guarantee that Y - A.Xapprox is perpendicular to the plane on which A hangs its hits.

And because A . Xapprox is on the plane on which A hangs its hits and because the perpendicular distance is the shortest distance, this tells you that

A.Xapprox is closer to Y than any other possible hit with A.

In short: Xsol is as close to a solution of:

```
linearsystem
```

$$-0.91\,x[1] + 0.83\,x[2] == -1.05$$
$$-0.62\,x[1] - 0.28\,x[2] == 0.35$$
$$-0.3\,x[1] + 1.2\,x[2] == -1.41$$

as you or anyone else will ever get.

That's why lost of folks call X the best approximate solution for this linear system.

□ B.4.a.iii) Other linear systems without exact solutions

Does the same thing work for other linear systems without solutions?

□ Answer:

You bet your sweet mouse it does!

As a matter of fact, it works for the same reasons examined above.

To whit:

When you analyze the coefficient matrix A, you get its rank and then you know that A hangs its hits on the part of hangdimD framed by

{hangerframe[1],hangerframe[2],,. . .,hangerframe[rank]}.

You put

$$\text{Xapprox} = \sum_{k=1}^{\text{rank}} \left(\frac{\text{Y.hangerframe[k]}}{\text{stretch[k]}}\right) \text{alignerframe[k]}$$

When you hit Xapprox with A you get

$$A.\,Xapprox = \sum_{k=1}^{rank} \left(\frac{Y.hangerframe[k]}{stretch[k]}\right) A.alignerframe[k]$$
$$= \sum_{k=1}^{rank} \left(\frac{Y.hangerframe[k]}{stretch[k]}\right) stretch[k]\, hangerframe[k]$$
$$= \sum_{k=1}^{rank} (Y.hangerframe[k])\;\; hangerframe[k]$$

So

$$Y - A.Xapprox =$$
$$Y - \left(\sum_{k=1}^{rank} (Y.hangerframe[k])\, hangerframe[k]\right).$$

Consequently for each p with $1 \le p \le$ rank, you get

$$(Y - A.Xapprox).hangerframe[p]$$
$$= (Y.hangerframe[p]) - \left(\sum_{k=1}^{rank} (Y.hangerframe[k])\,(hangerframe[k].)\, hangerframe[p]\right)$$
$$= (Y.hangerframe[p]) - (Y.hangerframe[p])\,(hangerframe[p].hangerframe[p])$$

This happens because hangerframe[k].hangerframe[p] = 0 when k and p are different.

$$= (Y.hangerframe[p]) - (Y.hangerframe[p])$$

This happens because hangerframe[p].hangerframe[p] = 1

$$= 0.$$

The upshot:

$$(Y - A.Xapprox).hangerframe[p] = 0$$

for all the individual vectors, hangerframe[p], in the hangerframe for A.

And this tells you that Y – A.Xapprox is perpendicular to the part of hangdimD on which A hangs its hits.

And because in all dimensions, the perpendicular distance is the shortest distance, this tells you that A.Xapprox is closer to Y than any other possible hit with A.

B.5) Inverses for invertible matrices and pseudoinverses for non-invertible matrices

If A = hanger.stretcher.aligner and A is invertible, then

A^{-1} = aligner^t unstretcher.hanger^t .

If A = hanger.stretcher.aligner, then a hit on Y with

PseudoInverse[A] = aligner^t unstretcher.hanger^t gives you Xapprox.

□ B.5.a.i) If A = hanger.stretcher.aligner, then a hit on Y with PseudoInverse[A] = aligner^t unstretcher.hanger^t gives you Xapprox

Explain this:
If A is any matrix and
A = hanger.stretcher. aligner,
then when you hit Y with

PseudoInverse[A] = aligner^t unstretcher.hanger^t

you get

$$Xapprox = \sum_{k=1}^{rank} \left(\frac{Y.hangerframe[k]}{stretch[k]}\right) alignerframe[k].$$

In other words, A.[PseudoInverse[A].Y] is as close to Y as any other hit with A.

□ Answer:

When you hit aligner^t unstretcher.hanger^t on Y, you get

aligner^t unstretcher.hanger^t.Y

$$= \text{aligner}^t\ \text{unstretcher}.\{Y.hangerframe[1], Y.hangerframe[2], \ldots, Y.hangerframe[rank]\}$$
$$= \text{aligner}^t.\left\{\frac{Y.hangerframe[1]}{stretch[1]}, \frac{Y.hangerframe[2]}{stretch[2]}, \ldots, \frac{Y.hangerframe[rank]}{stretch[rank]}\right\}$$
$$= \sum_{k=1}^{rank} \left(\frac{Y.hangerframe[k]}{stretch[k]}\right) alignerframe[k]$$
$$= Xapprox = PseudoInverse[A].Y$$

Try it out on a sample random matrix A

```
hitdim = Random[Integer, {3, 5}];
hangdim = Random[Integer, {4, 7}];

Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];

A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];

MatrixForm[A]
```

1.12367	0.987214	-0.413797	1.74803	-0.50687
0.696333	0.226752	-1.38017	0.277693	1.32287
-1.22926	-1.70456	1.45223	-0.239313	-0.994546
-0.703573	-1.10974	-1.71394	1.13896	-0.358381
1.34671	1.9013	0.425203	1.09607	-1.77696
-1.08591	-1.161	1.34804	0.729907	0.217753
0.612247	0.728216	-1.54779	0.89488	-0.158496

Here is aligner^t unstretcher.hanger^t:

This uses all the nonzero stretch factors and ignores any zero stretch factors.

```
rank = Length[SingularValues[A][[2]]];
Clear[alignerframe, k];
alignerframe[k_] := SingularValues[A][[3, k]];
aligner = Table[alignerframe[k], {k, 1, rank}];
Clear[stretch];
stretch[k_] := SingularValues[A][[2, k]];
stretcher = DiagonalMatrix[Table[stretch[k], {k, 1, rank}]];

Clear[hangerframe, k];
hangerframe[k_] := SingularValues[A][[1, k]];
hanger = Transpose[Table[hangerframe[k], {k, 1, rank}]];

unstretcher = DiagonalMatrix[Table[1/stretch[k], {k, 1, rank}]];
MatrixForm[Transpose[aligner].unstretcher.Transpose[hanger]]
```

0.788906	1.0335	1.06516	-0.300225	-0.0433188	-0.788082
-0.539201	-0.750686	-0.98888	0.000429602	0.177358	0.515837
0.150987	0.0560713	0.0927044	-0.257874	0.0394752	0.204782
0.225074	0.00351537	-0.147821	0.100872	0.0475978	0.445357
0.0307117	0.16049	-0.316242	-0.165284	-0.197857	0.342529

Compare with *Mathematica*'s built in instruction:

```
MatrixForm[PseudoInverse[A]]
```

0.788906	1.0335	1.06516	-0.300225	-0.0433188	-0.788082
-0.539201	-0.750686	-0.98888	0.000429602	0.177358	0.515837
0.150987	0.0560713	0.0927044	-0.257874	0.0394752	0.204782
0.225074	0.00351537	-0.147821	0.100872	0.0475978	0.445357
0.0307117	0.16049	-0.316242	-0.165284	-0.197857	0.342529

Rerun the last three cells a couple of times.

□ B.5.a.ii) If A = hanger.stretcher.aligner and A is invertible, then A^{-1} = PseudoInverse[A] = aligner^t unstretcher.hanger^t

Explain this:
If A is any invertible matrix and
A = hanger.stretcher. aligner,
then A^{-1}

$$A^{-1} = \text{PseudoInverse[A]} = \text{aligner}^t\ \text{unstretcher.hanger}^t.$$

□ Answer:

Go with a kD invertible matrix A and any vector Y in kD.

The solution of the linear system A.X = Y is

Xsol = A^{-1}.Y .

On the other hand,

Xappox = PseudoInverse[A].Y

is as close as it is possible to get to solution.

And because Xsol is the exact solution,

Xapprox = Xsol.

So

PseudoInverse[A].Y = A^{-1}.Y

for all Y's in kD.

This tells you that PseudoInverse[A] and A^{-1} are the same matrix.

Try it out on this sample invertible matrix:

This matrix hits on 5D and hangs in 5D.

```
hitdim = 5;
hangdim = 5;

Clear[a, i, j];
a[i_, j_] := Random[Real, {-2, 2}];
A = Table[a[i, j], {i, 1, hangdim}, {j, 1, hitdim}];

MatrixForm[A]
```

$$\begin{pmatrix} 0.432774 & -1.00002 & -0.865807 & -1.16395 & -0.863653 \\ -1.89027 & -1.15187 & -0.302908 & 1.49473 & -1.23698 \\ -1.05317 & 1.27189 & -1.60134 & -1.46002 & -1.96725 \\ 0.432889 & -0.949387 & -0.189928 & -0.185006 & 1.82064 \\ 0.322397 & -0.642142 & 0.920115 & -0.0208618 & 1.88962 \end{pmatrix}$$

Here's *Mathematica*'s calculation of A^{-1}

```
MatrixForm[Inverse[A]]
```

$$\begin{pmatrix} 0.154337 & -0.332892 & -0.468743 & 0.123747 & -0.754607 \\ -0.45679 & -0.215154 & 0.141767 & 0.0443645 & -0.244774 \\ 0.211895 & 0.018225 & -0.0396568 & -0.987852 & 1.01928 \\ -0.352755 & 0.0656761 & -0.372948 & 0.387105 & -0.879477 \\ -0.288634 & -0.024468 & 0.143343 & 0.479252 & 0.0687436 \end{pmatrix}$$

And here's *Mathematica*'s calculation of PseudoInverse[A]

```
MatrixForm[PseudoInverse[A]]
```

$$\begin{pmatrix} 0.154337 & -0.332892 & -0.468743 & 0.123747 & -0.754607 \\ -0.45679 & -0.215154 & 0.141767 & 0.0443645 & -0.244774 \\ 0.211895 & 0.018225 & -0.0396568 & -0.987852 & 1.01928 \\ -0.352755 & 0.0656761 & -0.372948 & 0.387105 & -0.879477 \\ -0.288634 & -0.024468 & 0.143343 & 0.479252 & 0.0687436 \end{pmatrix}$$

Rerun the last three cells a couple of times.

□ B.5.a.iii) Comparing inverse and pseudoinverse

When and how do you use the pseudoinverse instead of the inverse?

□ Answer:

You can use the inverse on matrices that are invertible.

You want to use the pseudoinverse on matrices that are not invertible.

You can use the pseudoinverse on invertible matrices too.
For invertible matrices, the inverse and the pseudoinverse are the same.

Here's a sample of what you can do with the pseudo inverse:

This matrix hits on 4D and hangs in 7D

```
A = {{0.45, 0.43, -1.06, -1.65},
     {-1.44, -1.28, 0.31, 1.60},
     {0.079, -1.96, 0.86, 1.73},
     {-0.57, 0.58, 1.82, -1.46},
     {1.39, 0.32, 0.44, -0.78},
     {1.89, -0.49, -1.12, 1.11},
     {-1.21, 0.55, 1.50, -0.64}};

MatrixForm[A]
```

$$\begin{pmatrix} 0.45 & 0.43 & -1.06 & -1.65 \\ -1.44 & -1.28 & 0.31 & 1.6 \\ 0.079 & -1.96 & 0.86 & 1.73 \\ -0.57 & 0.58 & 1.82 & -1.46 \\ 1.39 & 0.32 & 0.44 & -0.78 \\ 1.89 & -0.49 & -1.12 & 1.11 \\ -1.21 & 0.55 & 1.5 & -0.64 \end{pmatrix}$$

This matrix is not invertible:

```
Inverse[A]
```

```
Inverse::matsq : Argument {{0.45, 0.43, -1.06, -1.65}, <<5>>, {-1.21, 0.55, 1.5, -0.64}}
  at position 1 is not a square matrix.
Inverse[{{0.45, 0.43, -1.06, -1.65},
  {-1.44, -1.28, 0.31, 1.6}, {0.079, -1.96, 0.86, 1.73},
  {-0.57, 0.58, 1.82, -1.46}, {1.39, 0.32, 0.44, -0.78},
  {1.89, -0.49, -1.12, 1.11}, {-1.21, 0.55, 1.5, -0.64}}]
```

Yet there is no problem with the pseudoinverse of A:

```
MatrixForm[PseudoInverse[A]]
```

$$\begin{pmatrix} -0.0822778 & -0.181975 & 0.0893503 & 0.0525397 & 0.242858 & 0.212999 & -0 \\ -0.599057 & -0.196777 & -0.538087 & -0.12954 & -0.0501851 & 0.144852 & 0 \\ -0.253551 & -0.0725887 & 0.117975 & 0.205438 & 0.179309 & 0.0294268 & 0 \\ -0.547 & -0.0127467 & -0.193643 & -0.173591 & -0.0675111 & 0.189154 & 0 \end{pmatrix}$$

Example: Nailing a solution using the pseudoinverse

You can use PseudoInverse[A] to check whether the linear system A.X = Y has a solution for this 7D Y:

```
Y = {-6.4855, 5.2153, 8.63207, -4.7117, -0.9053, 5.6479, -2.9046};
Xapprox = PseudoInverse[A].Y
```

```
{1.23, -0.91, 0.87, 3.47}
```

Compare:

```
A.Xapprox
```

```
{-6.4855, 5.2153, 8.63207, -4.7117, -0.9053, 5.6479, -2.9046}
```

```
Y
```

```
{-6.4855, 5.2153, 8.63207, -4.7117, -0.9053, 5.6479, -2.9046}
```

The call:

Nailed an solution.

Example: Nailing a reasonably good approximate solution using the pseudoinverse

You can use PseudoInverse[A] to check whether the linear system A.X = Y has a solution for this new 7D Y:

```
Y = {-5.28, 2.44, 3.09, -5.24, -1.13, 5.35, -2.71};
Xapprox = PseudoInverse[A].Y
```

```
{0.985601, 1.9727, -0.0154578, 3.9889}
```

Compare:

```
A.Xapprox
```

```
{-5.27353, 2.43314, 3.09889, -5.26956, -1.1169, 5.34116, -2.68368}
```

```
Y
```

```
{-5.28, 2.44, 3.09, -5.24, -1.13, 5.35, -2.71}
```

The call:

Strictly speaking, there is no solution, but the approximate solution Xapprox is not all that bad.

Example: Using the pseudoinverse to see a linear system that is not even close to having a solution

You can use PseudoInverse[A] to check whether the linear system A.X = Y has a solution for this new 7D Y:

```
Y = {-4.28, 0.44, 6.09, -3.24, -0.13, -4.35, -0.71};
Xapprox = PseudoInverse[A].Y
```

```
{-0.278249, -1.14962, 0.844764, 0.834528}
```

Compare:

```
A.Xapprox
```

```
{-2.89197, 3.46932, 4.40151, -0.189119, -1.03388, 0.017614, 0.437438}
```

```
Y
```

```
{-4.28, 0.44, 6.09, -3.24, -0.13, -4.35, -0.71}
```

The call: Xapprox is the best that can be done. But the approximation sucks.

For this Y, the linear system A.X = Y is not even close to having a solution.

It's a sorry excuse for a linear system.

□ B.5.a.iii) An invertible matrix is a square matrix of full rank. Other matrices are not invertible

Explain these shots from the hip:

- If A is a matrix with hitdim < hangdim, then A cannot be invertible.
- If A is a matrix with hitdim > hangdim, then A cannot be invertible.
- If A is a matrix with hitdim = hangdim (i.e. A is a square matrix) , then saying that A is invertible is the same as saying that A is of full rank.

□ Answer:

▪ If A is a matrix with hitdim < hangdim, then A cannot be invertible.

Explanation: Hits with A can fill up at most hitdim dimensions of hangdimD. Because hitdim < hangdim, hits with A cannot fill all of hangdim D. There are at least hangdim-hitdim dimensions left out.

The upshot: There are many vectors Y in hangdim D that are not hits with A. You cannot undo A on these vectors. So A is not invertible.

▪ If A is a matrix with hitdim > hangdim, then A cannot be invertible.

Explanation: In this case, hits with A squash dimensions. There must be a vector Xsquash with $\|Xsquash\| \neq 0$ in hitdimD that a hit with A squashes to {0,0,...,0}.

But A.{0,0,...,0} = {0,0,...,0}.

So

A.Xsquash = A.{0,0,...,0} = {0,0,...,0}

If A is invertible, then

Xsquash = A^{-1}A.Xsquash = A^{-1}A.{0,0,...,0} = {0,0,...,0}.

This cannot happen because $\|Xsquash\| \neq 0$.

▪ If A is a matrix with hitdim = hangdim (i.e. A is a square matrix) , then saying that A is invertible is the same as saying that A is of full rank..

Explanation: If A is a matrix with hitdim = hangdim (i.e. A is a square matrix) , then saying that A is invertible is the same as saying that given a Y in handdimD, the corresponding linear system

AX = Y

is exactly determined.

This is the same as saying that A is of full rank.

B.6) Transposes in higher dimensions:

The rank of A is the same as the rank of A^t.

The nonzero stretch factors for A are the same as the nonzero stretchfactors for A^t

The non-zero aligner frame for A is the non-zero hangerframe for A^t.

The non-zero hanger frame for A is the non-zero aligner frame for A^t.

□ B.6.a.i) A test case

Here's a matrix A:

```
A = {{0.43, -1.32, -2.26, 1.47, 4.13, 2.32},
     {-0.34, 0.34, 0.00, 7.89, -2.96, 0.89},
     {2.58, 1.34, 3.82, 2.84, -1.44, 0.78},
     {0.61, 1.35, 1.91, 17.2, -6.64, 2.17},
     {1.72, -0.65, -0.35, 2.89, 3.41, 2.71}};
MatrixForm[A]
```

$$\begin{pmatrix} 0.43 & -1.32 & -2.26 & 1.47 & 4.13 & 2.32 \\ -0.34 & 0.34 & 0 & 7.89 & -2.96 & 0.89 \\ 2.58 & 1.34 & 3.82 & 2.84 & -1.44 & 0.78 \\ 0.61 & 1.35 & 1.91 & 17.2 & -6.64 & 2.17 \\ 1.72 & -0.65 & -0.35 & 2.89 & 3.41 & 2.71 \end{pmatrix}$$

This matrix hits on 6D and hangs in 5D.

Calculate the rank of A:

```
rankA = Length[SingularValues[A][[2]]]
```

```
3
```

Here's the transpose A^t of A:

```
MatrixForm[Transpose[A]]
```

$$\begin{pmatrix} 0.43 & -0.34 & 2.58 & 0.61 & 1.72 \\ -1.32 & 0.34 & 1.34 & 1.35 & -0.65 \\ -2.26 & 0 & 3.82 & 1.91 & -0.35 \\ 1.47 & 7.89 & 2.84 & 17.2 & 2.89 \\ 4.13 & -2.96 & -1.44 & -6.64 & 3.41 \\ 2.32 & 0.89 & 0.78 & 2.17 & 2.71 \end{pmatrix}$$

Calculate the rank of A^t:

```
rankAtrans = Length[SingularValues[Transpose[A]][[2]]]
```

```
3
```

The rank of $A^t$is the same of A.
Is this an just an accident?

□ Answer:

Uh-oh.

In mathematics, there are no accidents.

If you want to see why, then go on.

□ B.6.a.ii) Why rank of A = rank of A^t.

Explain this:
No matter what matrix A you go with, you are guaranteed that
rank of A = rank of A^t.

□ Answer:

Remember

A = hanger.stretcher.aligner.

So

A^t = $aligner^t$.$stretcher^t$.$hanger^t$

= $aligner^t$.stretcher.$hanger^t$

because, as a diagonal matrix, stretcher = $stretcher^t$.

This that you that A and A^t both use the same perpendicular frames for their hits.

Except the roles are reversed.

The perpendicular frame A uses for non-zero aligning is used by A^t for non-zero hanging.

The perpendicular frame A uses for non-zero hanging is used by A^t for non-zero aligning.

And they both have the same non-zero stretch factors.

That's why matter what matrix A you go with, you are guaranteed that

rank of A = rank of A^t.