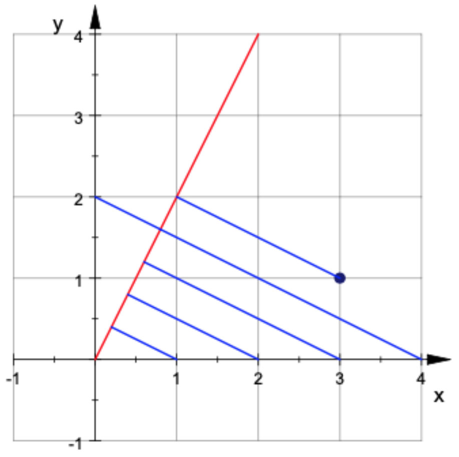
Exploring Math $\sum_{math}$ with EIGENMATH

Elementary Introduction to Pseudoinverse with Applications using Eigenmath

Overdetermined Systems • Best Fits • GREVILLE algorithm



Dr. Wolfgang Lindner
`LindnerW@t-online.de`
Leichlingen, Germany
2020

Contents

About this Booklet

Eigenmath

Eigenmath is a computer algebra system that can be used to solve problems in mathematics and the natural and engineering sciences. It is a personal resource for students, teachers and scientists. Eigenmath is small, compact, capable and free. It runs on WindowsOS, MacOS, Android and online in a browser. With the pseudoinverse, that was previously unused in elementary math, a tool is introduced and tested for the CAS Eigenmath in order to check its possibilities. After the introduction of the Moore-Penrose-inverse and its geometric interpretation as a factor of an orthogonal projection, theoretically and practical solutions between classic distance calculations and regression problems are possible. A characterizing definition for pseudoinverse is given and used to construct the explicit solution of systems of linear equations.

To the student

This booklet would like to accompany the reader to a high point of the matrix-oriented elementary linear algebra: *to the core idea of partial inversion of a matrix to a pseudoinverse.* In the case of over- or underdetermined or singular linear equation systems $A * X = B$, such 'pseudo'inverses allow the explicit, fully automated calculation of the solution set with only a few preconditions and allow a theoretically unified and practically powerful representation of the topic of solving Linear Systems. The general solution formula is an attempt to generalize the regular solution formula $X = A^{-1} * B$. The mental concept of the pseudoinverse consequently thinks the algebraic inversion idea for the solution of linear systems of equations to an closed end.
At the same time, algebraic and geometric insights are linked, since the special Moore-Penrose-inverse turns out to be geometrically a factor of an orthogonal projection and linear regression calculations are canonized and trivialized: e.g. the calculation of the regression line (parabola etc.) is compressed into a conceptual and CAS-explicit one-liner. The considerations made here would be difficult to elementarize without the use of a computer algebra system like Eigenmath because product formations of 3 to 5 matrices occur in the conceptual construction - with inversions inside. In Eigenmath laboratories we explore the decisive phenomena or verify or falsify hypotheses and would like to encourage ongoing dialogical practice in CAS language communication skills with the Eigenmath assistance.
Therefore the accompanying linguistic comments are deliberately short. If possible, all CAS dialog sequences - which are shown in `typewriter font` - should be performed live on the computer. If you pull a postcard in the Eigenmath input region going down step by step from an Eigenmath command to an indented answer in the Eigenmath output window (written in LaTeX) and allow yourself a short pause for a reflection, you can simulate this communication process in a rudimentary way - but it allows as a static reading act no spontaneous deviations, additional inquiries or desirable explorations, which is possible at the Eigenmath prompt region under the output window.

An interdisciplinary aspect occurs through the use of elementary methods of software engineering in the bottom-up development and step-by-step refinement of the functions `mpi` or `pinv` and `Greville`. Techniques of this kind can often be used in CAS and train algorithmic oriented constructive thinking. The EIGENMATH commands used and the textual representation should be elementary enough to serve as a companion while reading basic or advanced courses or as help system for independent individual work.

This small text[1] has fulfilled its purpose if the reader has learned to express himself in the mathematically-related symbolic CAS EIGENMATH-language as a mathematical language of communication and if he can use it to formulate problems as discussed here or to tackle own tasks in dialogue with the CAS EIGENMATH.

The mathematical requirements to read this text are minimal. A first course on elementary Linear Algebra should suffice. I recommend one of the books [2], [7][2], [9], [13][3] [19] or [20]. For an introduction to programming [18] is a good choice.

Any feedback from the user is very welcome.

PS: Being retired and no native speaker, I have no support from colleges at high school or university anymore, therefore the reader may excuse me for my grammatical and spelling mistakes.

Wolfgang Lindner
Leichlingen, Germany
December 2020

[1]This text is an enhanced version of [12], where the author used the CAS MuPAD.

[2]The use of DERIVE is a welcome opportunity to port the simple code to EIGENMATH.

[3]This is a fine tuned state-of-the-art introduction to LA, which uses a *Python* package for doing Geometric Algebra (GA). There is a corresponding package called `EVA` for EIGENMATH, which allows to follow the book with EIGENMATH, see url: `http://beyhfr.free.fr/EVA2/index.html`

1 First Steps towards the Moore-Penrose-Inverse

If a system of linear equations

$$A * X = B$$

is *regular*[4], then we know that one can find the unique solution X through the regular formula

$$X = A^{-1} * B$$

e.g. the system of linear equations is multiplied for a ('the unique') solution on both sides with the inverse matrix A^{-1}. In the natural, social and engineering sciences, linear systems of equations emerge frequently e.g. by measurements repeated at different times. The information obtained in this way is often subject to measurement errors and as a rule there are more or fewer equations ('informations') than unknowns. The resulting LS[5] are consequently over-determined or under-determined or, even worse, often not solvable at all. Therefore, in these cases one is dependent on the calculation of 'optimal' approximate solutions. For the concrete calculation of such best approximate solutions (often called 'best fit solutions'), we try to rescue the simple algebraic solution principle

$$A * X = B \Rightarrow X = [?] * B$$

as much as possible, i.e. we are looking for a meaningful 'substitute' matrix [?] for the non-existent inverse matrix. This is the aim of the first chapter.

1.1 Insolvable Linear Systems and their 'proximal' Solutions

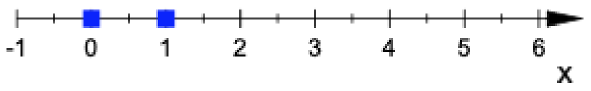
At the beginning we want to train our intuition for the selection of 'elements of best fit' - one also speaks of 'best compromise solutions' - for *un*solvable LS.

1.1.1 *Exercise:* Insolvable Linear Systems and their 'best fit' solutions

Look for best approximate solutions in the following problems.

a) to 'solve' an unsolvable LS:

$$\{x = 0,\ x = 1\}$$



An unsolvable 'overdetermined' linear System of equations: 2 equations for 1 unknown x.

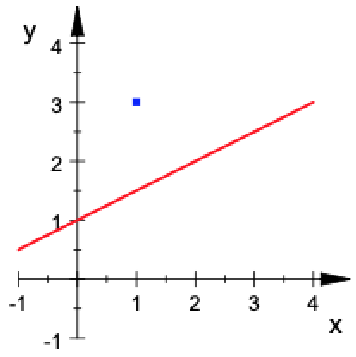
What is your guess of a ('the' ?) optimal compromise solution? Can you see it?

[4]e.g. unique solvable, that means the determinant of A is non-zero.

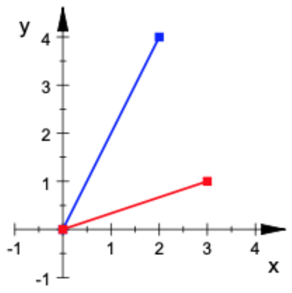
[5]LS = Linear System : that is a set or collection of individual linear equations. Think of the two equations $x + y = 1,\ x - y = 0$ as an example of a 'system'.

b)

Estimate the distance of the point from the straight line, do not calculate.



c) 'solve' another unsolvable LS per intuition.



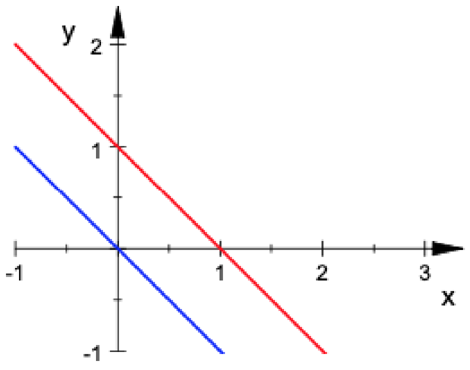
$$2 \cdot x = 3$$
$$4 \cdot x = 1$$

Why unsolvable?
Why overdetermined?
Interpretation of the LS?

Propose an ('the') optimal compromise solution.

d) Parallel straight lines:

$$x + y = 0$$
$$x + y = 1$$

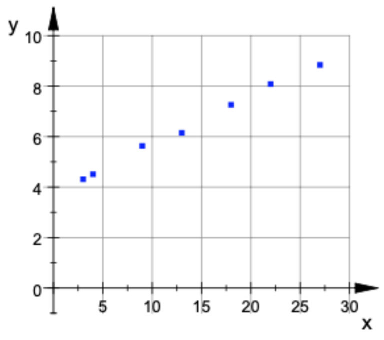


Why unsolvable? Overdetermined?

Is there an ('the') optimal compromise solution? Use a short plea to argue.

e) Weight gain:

Woche	*kg*
3	4.31
4	4.51
9	5.63
13	6.15
18	7.26
22	8.08
27	8.84



From a table of the weight gain of an infant ...

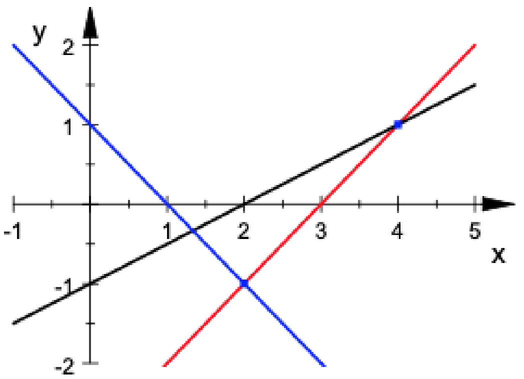
... the best possible estimate should be argued for his mean weight gain per week. Make a straight line through the point cloud by eye and estimate the weight in the 30th week (=Woche in German).

f) Bundle of straight lines:

$$x + y = 1$$

$$x - y = 3$$

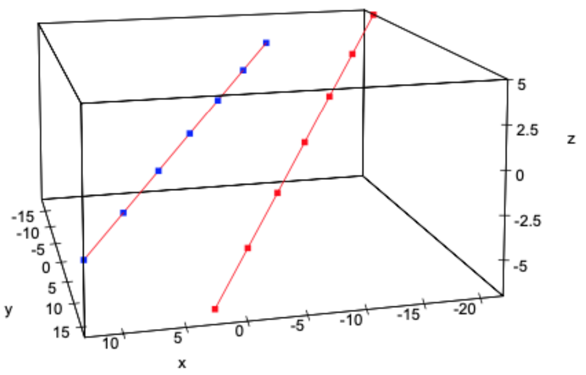
$$2 \cdot y - x = -2$$



Unsolvable? Overdetermined?

Can you locate an ('the only') best fit solution?

g) crooked straight lines:



Unsolvable? Overdetermined?

To which problem could a solution be sought of? Can you justify a best fit solution? Can you roughly estimate this solution from the drawing?

h) Translate the linear systems of equations from a) to g) in matrix form $A * X = B$.

If a linear system of equations $A * X = B$ is *inconsistent*[6], then in practically important cases we look for a way to nevertheless find a best fit solution. So instead of just giving up when we know that an LS is unsolvable, we compromise and try to find a vector X_o such that $A * X_o$ is at least as close as possible to B: we write

$$A * X_o \approx B$$

Such an X_o, which fulfills the equation $A * X = B$ in the best approximation, is called a *best fit* solution, the associated vector $B_o := A * X_o$ is a best approximation to B or a *proximum*[7] for B. Summarized:

best fit solution   target value

$$\downarrow \quad \downarrow$$

$$\underbrace{A * X_o}_{=:B_o} \approx B$$

$$\uparrow$$

proximum for B

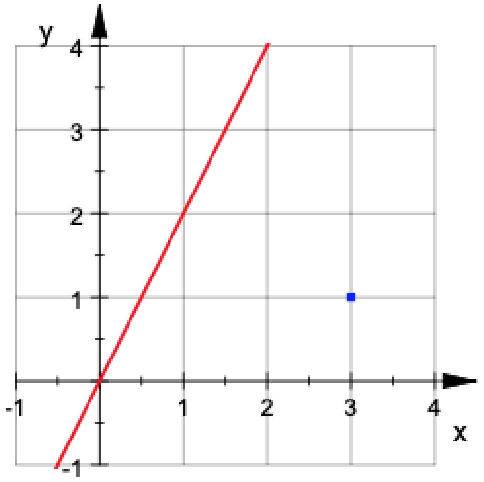
Hint. Here are some solution hypotheses for Ex.1.1.1, presented by
Adam: in a) I guess $x = 0.5$ as this is exactly between the two numbers 0 and 1.
Berta: I estimate the distance at about 1.5.
Carola: doesn't know whether she should answer $(2.5; 2.2)$ or $\frac{1}{2}$.
Who can help with an argument?
David: thinks that this can only be the central parallel $x + y = \frac{1}{2}$.
Erika: tells us her solution to e) later .. .
Fred: advocates for a certain point in the enclosed triangle - maybe the center point .. but he didn't find a clear hypothesis .. hm.
George: says you can't see anything because .. but there could be a connection with .. wait!

[6] i.e. is not solvable
[7] proximum (Latin: the closest [element]); in linguistic analogy to familiar concept of a maximum

We begin our investigations with the solution of Ex.1.1.1.c. To do this, we first make some preparations, including a measure for the optimality of approximate solutions.

1.1.2 *Exercise:* **Length and Distance**

What is the distance between the point $R(3;1)$ and the straight line with the equation $g\colon y = 2x$?



An visual estimate for the distance from R to g is ≈ 1.8 cm. To solve this problem arithmetically, we need
- a measure for calculating the distance between two points and
- a method for calculating the minimal distance from R to g.

a) Calculate by paper and pencil how far the point $R(3;1)$ is from the origin $N(0;0)$.

b) Calculate the distance from point $R(3;1)$ to point $Q(2;4)$.

c) What is the distance of the point $R(x,y)$ to the origin $N(0,0)$.

d) Argue: the distance between two points $P(x,y)$ and $Q(v,w)$ is $\sqrt{(x-v)^2+(y-w)^2}$

This motivates the following

1.1.3 *Definition:* **Length and Distance**

Consider two vectors ('points') $P = (P_1, P_2, ..)$ and $Q = (Q_1, Q_2, ..)$. The real number

$$|P| \overset{\text{def}}{=} \sqrt{P \bullet P} = \sqrt{P_2 + P_2 + ...} \tag{1.1}$$

is called the *length* or the *absolute value* of P and the real number

$$\text{dist}(P,Q) \overset{\text{def}}{=} |P-Q| = \sqrt{(P-Q)^2 + (P-Q)^2 + ...} \tag{1.2}$$

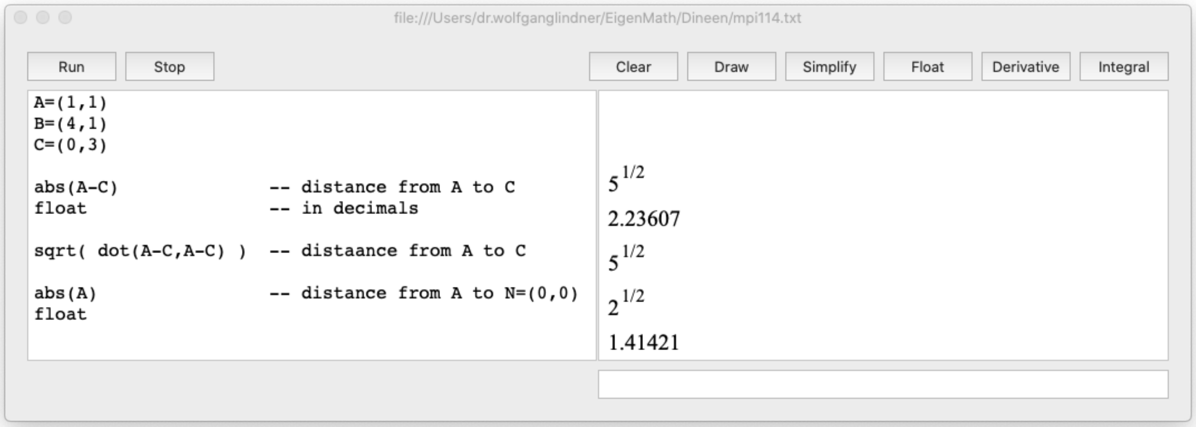
the distance between P and Q. The $\bullet$ in (1.1) denotes the scalarproduct of P and Q. With these two formulas length and distance are calculated by means of $|...|$.

Remark. Often, instead of $Q - P$, one also writes $\vec{PQ}$ and calls it the *direction vector* from P to Q. The direction vector from P to Q is therefore nothing but the difference $Q - P$, e.g. $\vec{PQ} \overset{\text{def}}{=} Q - P$. In this interpretation, $dist(P, Q) = |P - Q| = |\vec{PQ}|$ is the length of the direction vector $\vec{PQ}$.
In the following, we mostly use the explicit representation $Q - P$ for calculations and transformations, but often speak $Q - P$ as 'vector from P to Q', in calculations also as 'Q minus P'.

1.1.4 Eigenmath: Length and Distance

Calculate the distances of the corner points to the zero point and the side lengths in the triangle ABC with the points
a) $A = (1; 1)$, $B = (4; 1)$, $C = (0; 3)$ in $\mathbb{R}^2$
b) $A = (1; 1; 2)$, $B = (4; 1; 0)$, $C = (0; 3; 3)$ in $\mathbb{R}^3$ with and without Eigenmath.

Solution. Here is a start with a fresh Eigenmath session on the iMac ...


file:///Users/dr.wolfganglindner/EigenMath/Dineen/mpi114.txt
Run
Stop
Clear
Draw
Simplify
Float
Derivative
Integral

```
A=(1,1)
B=(4,1)
C=(0,3)

abs(A-C)                -- distance from A to C
float                   -- in decimals

sqrt( dot(A-C,A-C) )    -- distaance from A to C

abs(A)                  -- distance from A to N=(0,0)
float
```

$5^{1/2}$
2.23607
$5^{1/2}$
$2^{1/2}$
1.41421

For the handling of the inputs and outputs in the Eigenmath windows please consult the Eigenmath manual.

Lexicon: Math vs. Eigenmath

	mathematical notation	Eigenmath notation
1	the *norm* or *length* of matrix A, written $\| A \|$ or $\|A\|$	`abs(A)`
2	the squareroot of real number r, e.g. $\sqrt{r}$	`sqrt(r)`
3	the scalarproduct (or dotproduct) of vectors A and B, e.g. $A \bullet B$ or $< A, B >$	`dot(A,B)`
4	the distance from P to Q, e.g. dist(P, Q)	`dist(P,Q) = abs(P-Q)`

1.1.5 Eigenmath Lab: Measuring the optimality of approximate solutions

Consider the following unsolvable system of linear equations $A * X = B$ in matrix form, which we formulate inside the Eigenmath input region (left window)[8]:

$$\begin{pmatrix} 2\cdot x+y \\ -x+2\cdot y \\ x+2\cdot y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

We try to get closer to the best fit solution by 'trying to shoot' at target B with the help of Eigenmath. Here are three shot attempts $X1, X2, Xo$:

Eigenmath user input:[9]

```
-- seaching for best fits
A=((2,1),(-1,2),(1,2))
A
X=(x,y)
B=(1,2,3)
B
dot(A,X) -- left side of LS is A*X

X1=(1,2)
B1=dot(A,X1) -- B1=A*X=(4,3,5)
B1

X2=(1,0)
B2=dot(A,X2) -- B2=A*X=(2,-1,1)
B2

Xo=(0.1,1.2) -- Xo is a good fit
Bo=dot(A,Xo) -- Bo=(1.4,2.3,2.5)
Bo
```

Eigenmath output in LaTeX form:

$$\begin{bmatrix} 2x + y \\ -x + 2y \\ x + 2y \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

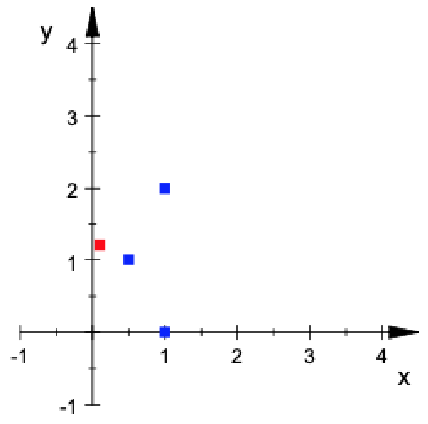
$$B_o = \begin{bmatrix} 1.4 \\ 2.3 \\ 2.5 \end{bmatrix}$$

We look graphically at the 2-dimensional candidates for best fit solutions Xi of the system of equations $A * X = B$ and the corresponding 3D approximations (proxima) Bi to B (in the pictures there is a fourth test point $X3 = (0.5; 1)$ with value $B3 = (2; 1.5; 2.5)$ drawn):

[8]The following left window (framed box) is not a screenshot, so you can copy and paste its contents to the real Eigenmath input wwindow

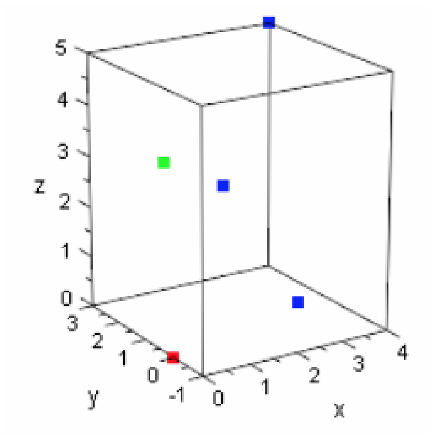
[9]If you are not on your iMac, please use the Eigenmath *online demo* [21] in your favorite browser, which is available on every operation system. Please copy or write these input commands into the Eigenmath browser window.

Test inputs of the user:



Protocol of the four 2D bet fit candidate solutions Xi of the LS $A * X = B$; the 'Sunday' shot X_o is marked in red.

Corresponding calculated values:



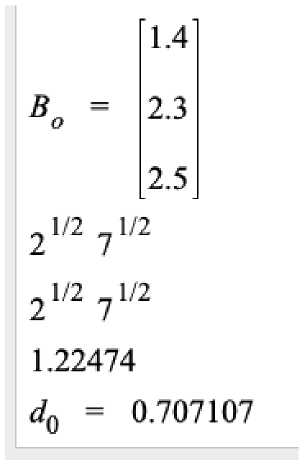
Looking at the corresponding 3D approximations (proxima) Bi to B. The origin (zero point) $N = (0; 0; 0)$ of the coordinate system is marked in red, the 'target' point B in green.

The smaller the distance of Bi to B the better fits Bi to B. So we look for the smallest value of dist(Bi, B) with the help of EIGENMATH:

EIGENMATH user input:

```
B=(1,2,3) -- target point e.g.  RHS
of LS
B1=(4,3,5)
B2=(2,-1,1)
B3=(2,1.5,2.5)
Bo=(1.4,2.3,2.5)
Bo

dist(P,Q) = abs(P-Q)
dist(B1,B)
dist(B2,B)
dist(B3,B)
d0=dist(Bo,B)
d0
```

EIGENMATH output:



$$B_o = \begin{bmatrix} 1.4 \\ 2.3 \\ 2.5 \end{bmatrix}$$

$$2^{1/2}\ 7^{1/2}$$

$$2^{1/2}\ 7^{1/2}$$

$$1.22474$$

$$d_0 = 0.707107$$

Result: for the moment $Bi = Bo$ is the best fit to B and is currently our proximum. We have $Bo = (1.4; 2.3; 2.5) \approx (1, 2, 3) = B$.
Try to get closer to B!

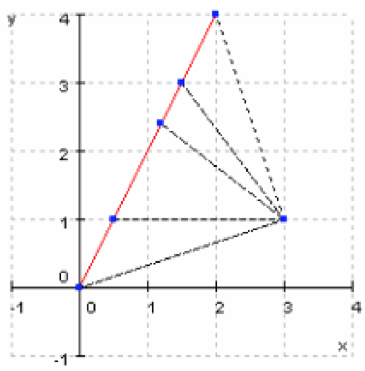
1.1.6 Definition: Goodness of Approximation - best fit solution

Xo is a *best fit solution* of $A * X = B$ if and only if

$$\mathrm{dist}(A * Xo, B) \leq \mathrm{dist}(A * X, B) \tag{1.3}$$

for all X.

Remark.



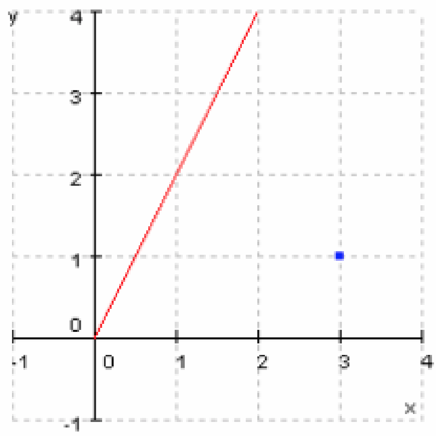
In the picture the $Bi := A * Xi$ are points on the red line, B is the single point outside the line. Interpret the best fit condition via the sketch and locate the proximum $Bo := A * Xo$.

Process planning. Develop as many different solution ideas as possible to solve the following problem: What is the distance between the point $R(3; 1)$ and the straight line with the equation g: $y = 2x$ from Ex.1.1.1.. Carry out a plan and check the result on the picture.

1.1.7 EIGENMATH Lab: solution of Ex.1.1.2 using an orthogonal projection

We now solve the point-line distance problem of Ex.1.1.2 with the help of an orthogonal projection. We have to answer the question: *What is the distance between the point $R(3; 1)$ and the straight line with the equation g: $y = 2x$?*
We formulate an idea for a solution in four different ways:

1. as a **picture** to get a visualisation



Here we get the impression that the solution should be ca. 2 cm.

2. in **words** for understanding:

   the distance from point R to g is the length of the perpendicular from R to g, so dist $(R, g) = |R - F| =$ 'length of the vector from F to R' with $F \in g$ as cut point from g and the perpendicular. Since we know R, we have to compute F and then we can calculate $|R - F|$ according to formula (1.2).

3. as an **equation** for doing arithmetic:

   Point F results from 2 conditions that we formulate as equations:

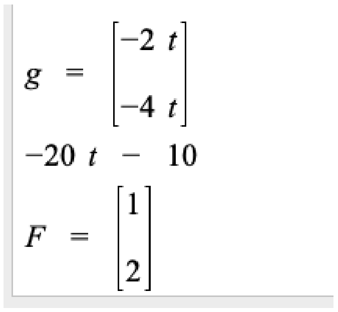
   (1) $R - F \perp P - Q$ e.g. $(R - F) \bullet (P - Q) = 0$
   (2) $F \in g$ e.g. $F = P + t(P - Q)$ with $t \in \mathbb{R}$ and $P, Q \in g$

   Instead of F, we now have to calculate the unkown number t from the determining equation (1), if we eliminate the unknown point F in (1) using (2).

4. using EIGENMATH as an math engine to help during the calculation:

EIGENMATH user input: EIGENMATH output:

```
R=(3,1)
P=(0,0)
Q=(2,4) -- because g=g(P,Q)
g=P+t*(P-Q)
g
dot(R-g,P-Q) -- we conclude t = -1/2

F=eval(g,t,-1/2)
F
```



$$g = \begin{bmatrix} -2\ t \\ -4\ t \end{bmatrix}$$

$$-20\ t - 10$$

$$F = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

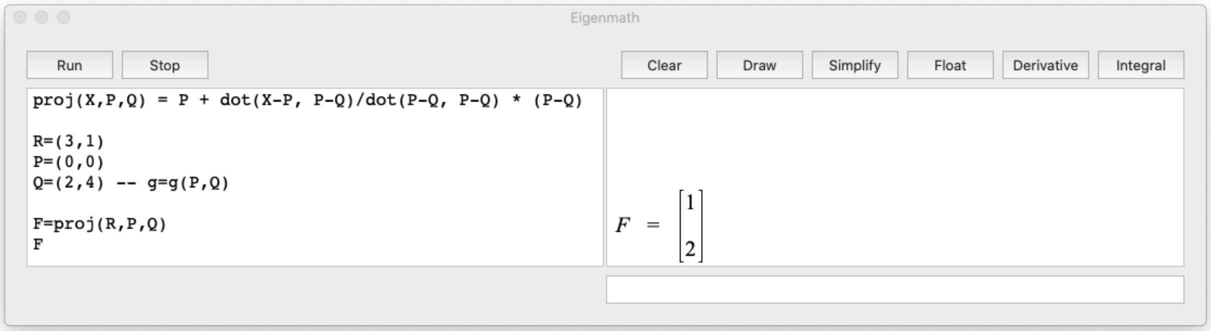
Result: point $F(1, 2)$ is the proximum, which is taken for the best fit value $t = 1/2$. The distance from R to g is therefore `|R-F|` `=abs(R-F)` $= \sqrt{5} \approx 2.24$.

Recapitulate the solution procedure in a few sentences in your own words. Could the solution process be automated? Make a test of the result on the sketch. Why is F the sought-after proximum from 1.1.2? What is the cause of optimality?

Surplus: We are not fully confident. We want to have an automatic calculation of F! Luckily from (1) and (2) it follows: $((R - P) - t(P - Q)) \bullet (P - Q) = 0$, so we can solve for t and calculate the base point F as follows

$$F \overset{\text{def}}{=} P + \frac{(X - P) \bullet (P - Q)}{(P - Q) \bullet (P - Q)} * (P - Q) \tag{1.4}$$

We use eq. (1.4) to immediately define an automatic EIGENMATH procedure *proj*(X, P, Q) and try it out with our example.
5. a fully automatic solution using EIGENMATH.



```
proj(X,P,Q) = P + dot(X-P, P-Q)/dot(P-Q, P-Q) * (P-Q)

R=(3,1)
P=(0,0)
Q=(2,4) -- g=g(P,Q)

F=proj(R,P,Q)
F
```

$$F = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Remark. Instead of the so called *projection formula*, which we used in the EIGENMATH session in 5. above

```
proj(X,P,Q) = P + dot(X-P, P-Q)/dot(P-Q, P-Q) * (P-Q)
```

one can equally use the following formula for defining the EIGENMATH procedure `proj`:

```
proj(X,P,Q) = P + dot(X-P, P-Q)/abs(P-Q)^2 * (P-Q)
```

1.1.8 EIGENMATH Lab: solution of an overdetermined Linear System

We now treat Ex.1.1.1.c and 'solve' the unsolvable LS $\{2x = 3, 4x = 1\}$.

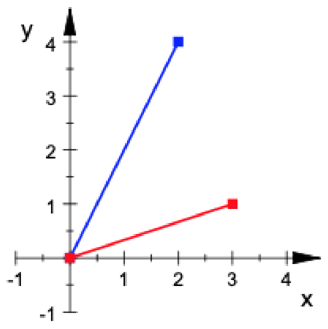


Figure 1: an overdetermined system of 2 linear equations for 1 unkown x.

This LS is overdetermined, there are 2 *contradicting* equations for 1 unknown x, therefore the solution set is empty $\{\}$. We reformulate the problem with matrices and start an attempt for an matrix-algebraic solution. We choose $A = \binom{1}{2}$, $B = \binom{3}{1}$ and $X = (x)$, so the LS is equivalent represented as matrix equation. We have:

$$\begin{aligned} 2x = 3 \quad \text{and} \quad 4x &= 1 \\ \begin{pmatrix} 2 \\ 4 \end{pmatrix} \star (x) &= \begin{pmatrix} 3 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 2x \\ 4x \end{pmatrix} &= \begin{pmatrix} 3 \\ 1 \end{pmatrix} \end{aligned}$$

Core idea: $A = \binom{2}{4}$ is rectangular, so not square, so we can't invert A, that is A^{-1} does not exist e.g.

$$A * X = B \not\Rightarrow X = A^{-1} * B$$

So: no chance for a 'standart' solution. We are therefore try to 'squareshape' A, e.g. to transform A in an $n \times n$ - matrix shape. We find the same numbers of A in a mirrored arrangement in their so-called *transposed* matrix A^t and we can always form the product $A^t * A$ ('squareshape'ing' A) . The new smaller 'squareshaped' LS for the unknowns x and y is formed trough multiplying the original matrix equation by A^t on both sides to get the so-called *normalequation*

$$A^t * A * X = A^t * B \tag{1.5}$$

which we can try to solve by inversion because in our case $(A^t * A)^{-1} = (1/20)$[10]

$$(20) \star (x) = (10)$$

therefore[11]

$$(x) = (1/20) \star (10)$$

$\therefore$

$$(x) = (1/2)$$

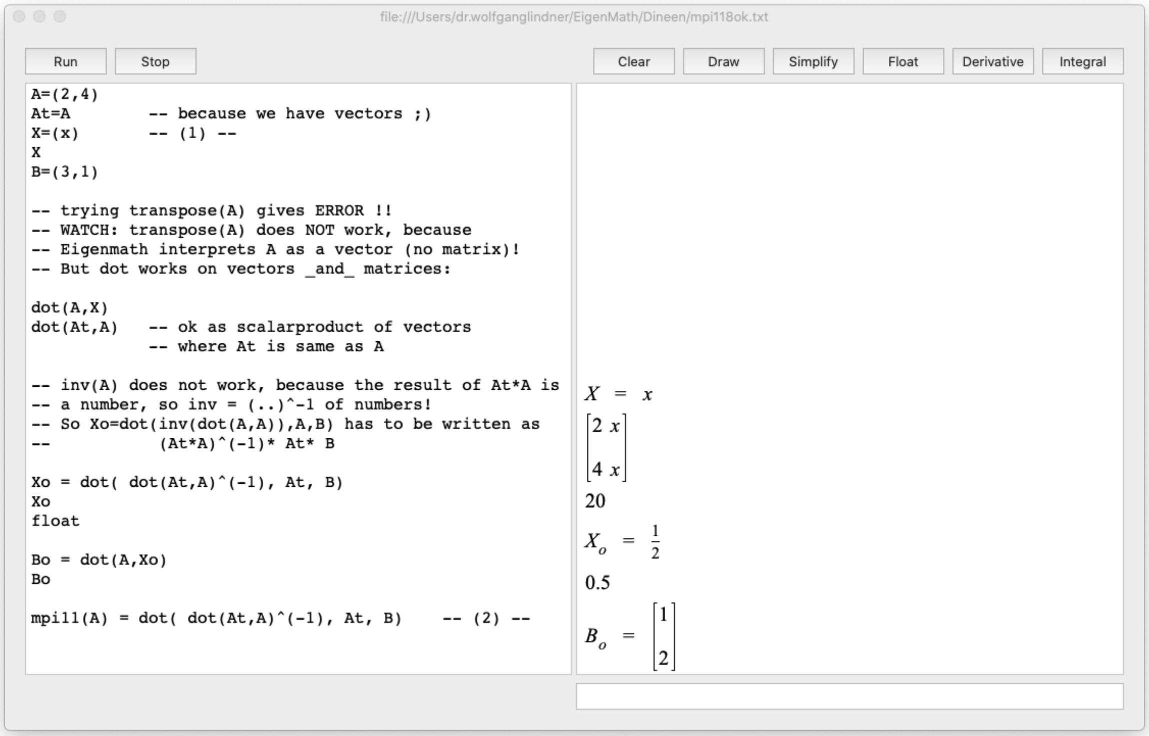
In summa we have computed $X = (x)$ via isolating X at the left side of equation (1.5):

$$X = (A^t * A)^{-1} * A^t * B \tag{1.6}$$

We now follow this recipe using EIGENMATH!

[10]Watch the difference between Math and EIGENMATH: In our case the matrix $A^t * A$ is the $1x1$-matrix (20) with the number 20 as single entry. Here $*$ denotes the matrix multiplication of Linear Algebra giving *matrices* as result. In contrast, if A and B are vectors then their multiplication is build via the dotproduct $\bullet$ of Linear Algebra giving a *real number* as result. WARNING: *in* EIGENMATH *there is only one product for vectors and matrices* alike named `dot( )`! So using EIGENMATH one has to be penible in respect with the result of the multiplication. This is to be observed in the following session.

[11]$A \therefore B$ means that A is true, and therefore B is true.


file:///Users/dr.wolfganglindner/EigenMath/Dineen/mpi118ok.txt
Run
Stop
Clear
Draw
Simplify
Float
Derivative
Integral

```
A=(2,4)
At=A        -- because we have vectors ;)
X=(x)       -- (1) --
X
B=(3,1)

-- trying transpose(A) gives ERROR !!
-- WATCH: transpose(A) does NOT work, because
-- Eigenmath interprets A as a vector (no matrix)!
-- But dot works on vectors _and_ matrices:

dot(A,X)
dot(At,A)   -- ok as scalarproduct of vectors
            -- where At is same as A

-- inv(A) does not work, because the result of At*A is
-- a number, so inv = (..)^-1 of numbers!
-- So Xo=dot(inv(dot(A,A)),A,B) has to be written as
--          (At*A)^(-1)* At* B

Xo = dot( dot(At,A)^(-1), At, B)
Xo
float

Bo = dot(A,Xo)
Bo

mpi11(A) = dot( dot(At,A)^(-1), At, B)    -- (2) --
```

$X = x$

$\begin{bmatrix} 2x \\ 4x \end{bmatrix}$

20

$X_o = \frac{1}{2}$

0.5

$B_o = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Remark. Here are some comments about the last session. In (1) we set `X=(x)`, but looking at the output we see that EIGENMATH has interpreted the input as a *number* - giving x back without the matrix (..) bracket! Therefore we can not build the transpose matrix A^t and `dot(At,A)` is given back as a *number* - being the result of a scalarproduct of two vectors. Consequently we have to 'invert' the *number* `dot(At,A)` which is to be done by means of the reciproc, noted `1/..` $= \ldots^{-1}$. This is shown in input (2).

Result. The unsolvable LS has a best fit solution $Xo = 1/2$ with the proximum $B0 = dot(A, Xo) = (1; 2)$. Looking at figure p.14 it seems that Bo is the base point of the orthogonal projection of B onto the line $g((0, 0), A)$.

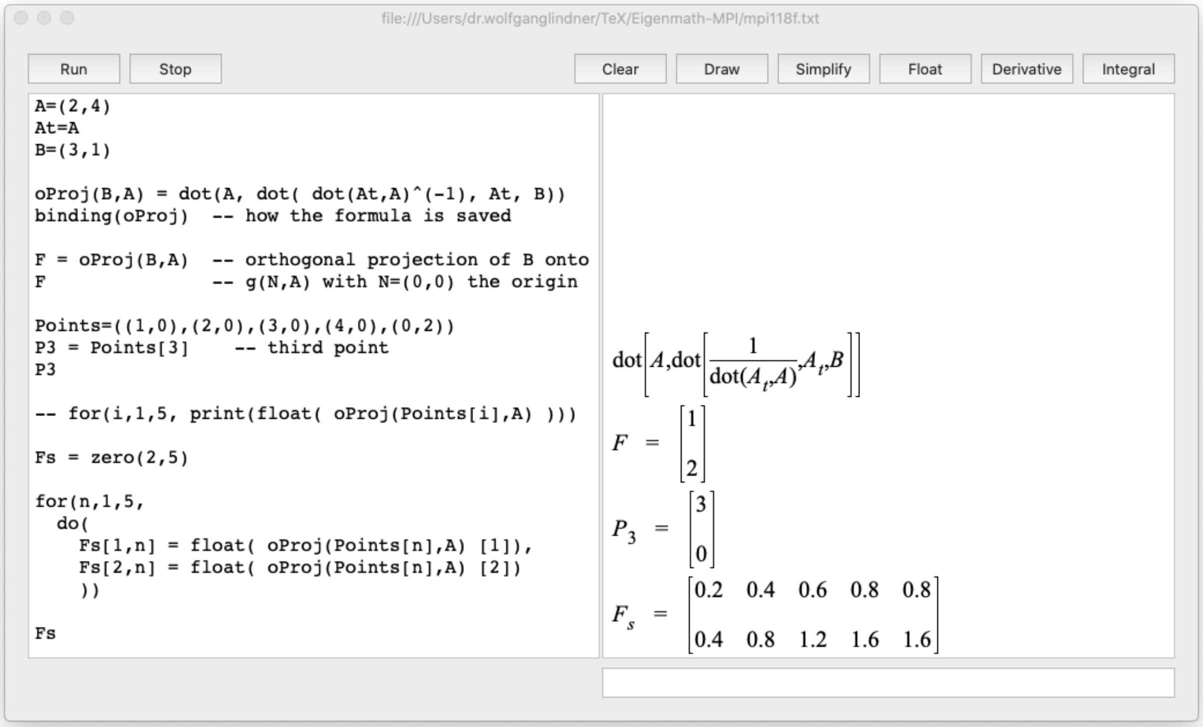
• Verify this hypothesis using e.g. formula (1.4).

Verification. Finishing our exploration we start a new lab and use formula (2) from the last session to verify our hypothesis about the geometric interpretation of `mpi11` as an orthogonal projection. Therefore we define:

```
oProj(B,A) = dot(A, dot( dot(At,A)^(-1), At, B))
```

Then we collect 5 points from the x-axis in a list (matrix) named `Points` and calculate their values by `oProj`, collecting the coordinates of the image points in the list (matrix) `Fs`[12].

[12]If one is only interested in viewing the coordinates of image points one can use the easier command `for(i,1,5, print(float( oProj(Points[i],A) )))`


file:///Users/dr.wolfganglindner/TeX/Eigenmath-MPI/mpi118f.txt
Run
Stop
Clear
Draw
Simplify
Float
Derivative
Integral

```
A=(2,4)
At=A
B=(3,1)

oProj(B,A) = dot(A, dot( dot(At,A)^(-1), At, B))
binding(oProj)  -- how the formula is saved

F = oProj(B,A)  -- orthogonal projection of B onto
F               -- g(N,A) with N=(0,0) the origin

Points=((1,0),(2,0),(3,0),(4,0),(0,2))
P3 = Points[3]     -- third point
P3

-- for(i,1,5, print(float( oProj(Points[i],A) )))

Fs = zero(2,5)

for(n,1,5,
  do(
    Fs[1,n] = float( oProj(Points[n],A) [1]),
    Fs[2,n] = float( oProj(Points[n],A) [2])
    ))

Fs
```

$$\mathrm{dot}\left[A,\mathrm{dot}\left[\frac{1}{\mathrm{dot}(A_t,A)},A_t,B\right]\right]$$

$$F = \begin{bmatrix}1\\2\end{bmatrix}$$

$$P_3 = \begin{bmatrix}3\\0\end{bmatrix}$$

$$F_s = \begin{bmatrix}0.2 & 0.4 & 0.6 & 0.8 & 0.8\\0.4 & 0.8 & 1.2 & 1.6 & 1.6\end{bmatrix}$$

If we plot the columns of Fs as points in the plane in Fig. p.14 we get the following graphic:

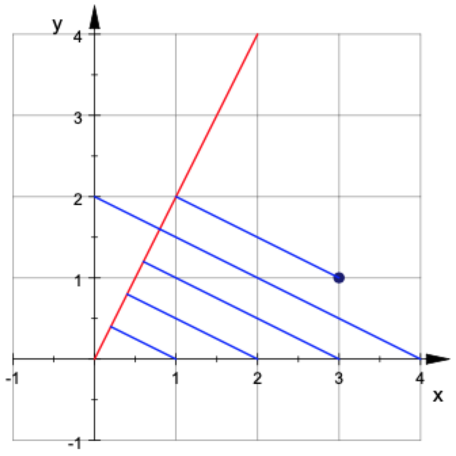


Figure 2: the algebraic function mpi.. acts as an orthogonal projection.

This confirms our hypothesis about the geometric background of the algebraic construction mpi11[13] as an orthogonal projection.

[13]The first 1 in the notation mpi11 reminds at the condition that the crucial matrix At*A must be invertible. The second 1 reminds that this is a special version of the construction mpi.. which is only valid for 1-dimensional matrices e.g. vectors.

1.1.9 Eigenmath Lab: Proximum of a bundle of straight lines

We are now testing the new concepts for calculating a best fit solution Xo and the associated proximum Bo on Ex.1.1.1.f, the 'bundle of lines'. The starting point is here the unsolvable overdetermined system of 3 equations for the 2 unkowns x and y:

$$\begin{aligned} x+y &= 1 \\ x-y &= 3 \\ -x++2y &= -2 \end{aligned}$$

We reformulate the problem with matrices to start an attempt for a matrix-algebraic solution:

$$\overset{A}{\begin{pmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 2 \end{pmatrix}} \overset{X}{\begin{pmatrix} x \\ y \end{pmatrix}} = \overset{B}{\begin{pmatrix} 1 \\ 3 \\ -2 \end{pmatrix}}$$

Main obstacle: A is rectangular, so not square, so we can't invert A, that is: there does not exist the matrix inverse of A. So: once again no chance for a 'standart' solution. Nevertheless we are able to bulid the transposed matrix A^t and we can form the *crucial* product $A^t * A$. The new smaller quadratic LS for the unknowns x and y is gained by considering the *normalequation*:

$$A^t * A * X = A^t * B \tag{1.7}$$

or in this concrete example

$$\begin{pmatrix} 3\cdot x - 2\cdot y \\ -2\cdot x + 6\cdot y \end{pmatrix} = \begin{pmatrix} 6 \\ -6 \end{pmatrix}$$

therefore

$$\begin{aligned} 3x-2y &= 6 \\ -2x+6y &= -6 \end{aligned}$$

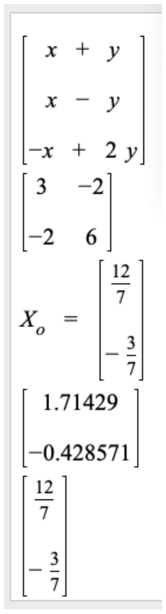
Because $At * A = \left(\begin{smallmatrix} 3 & -2 \\ -2 & 6 \end{smallmatrix}\right)$, $At * B = \left(\begin{smallmatrix} 6 \\ -6 \end{smallmatrix}\right)$ and $\det\left(\begin{smallmatrix} 3 & -2 \\ -2 & 6 \end{smallmatrix}\right) = 22 \neq 0$, we can solve this reduced LS by inversion to get

$$X = (A^t * A)^{-1} * A^t * B \tag{1.8}$$

The carry out this calculation along formula (1.8) with Eigenmath!

EIGENMATH user input: EIGENMATH output:

```
A=((1,1),(1,-1),(-1,2))
X=(x,y)
B=(1,3,-2)

At=transpose(A)

dot(A,X)
dot(At,A)

Xo=dot(inv(dot(At,A)),At,B)
Xo
float

mpi1(A)=dot(inv(dot(transpose(A),A)),transpose(A))

dot(mpi1(A),B)
```



$$\begin{bmatrix} x + y \\ x - y \\ -x + 2y \end{bmatrix}$$

$$\begin{bmatrix} 3 & -2 \\ -2 & 6 \end{bmatrix}$$

$$X_o = \begin{bmatrix} \frac{12}{7} \\ -\frac{3}{7} \end{bmatrix}$$

$$\begin{bmatrix} 1.71429 \\ -0.428571 \end{bmatrix}$$

$$\begin{bmatrix} \frac{12}{7} \\ -\frac{3}{7} \end{bmatrix}$$

Result: for the best fit $Xo = (12/7; -3/7)$ we have the proximum $Bo = (9/7; 15/7; -18/7) \approx (1; 3; -2) = B$. But the result Xo cannot easily be interpreted in the figure from Ex.1.1.1.f. Can we find a geometric interpretation of the solution analogous to the one in section 1.1.9? Investigate whether there is some kind of projection acting in the background, e.g. whether anything is anyhow projected to somewhere'.

$\Diamond$

Motivation. For an e.g. overdetermined unsolvable LS $A * X = B$ we had bulid the compensation solution

$$Xo = (A^t * A)^{-1} * A^t * B$$

or in EIGENMATH notation

```
(+)          Xo = dot(inv(dot(transpose(A),A)),transpose(A),B)
```

If we mentally compress the subterm $(A^t * A)^{-1} * A^t$ into a new symbol A^+, the compact solution term is now $Xo = A^+ * B$ and we consider

$$A^+ \overset{\text{def}}{=} (A^t * A)^{-1} * A^t$$

as the replacement for the missing inverse A^- of A. This leads us to the following definition of a new mathematical concept: the MOORE-PENROSE-*Pseudoinverse*, in short the `mpi`.

1.1.10 Definition: the Moore-Penrose-*Pseudoinverse*

Our algebraic key idea in the previous problems was: linear equations $A * X = B$ with a rectangular system matrix A are converted into a new equation system with a square system matrix $A^t * A$ by multiplication with the transpose A^t and then hopefully solved. We received a two-step solution method:

$$A*X=B$$

with A not invertible
e.g. $\det(A) = 0$ or $A \neq \square$

❶ $\downarrow$

$$A^t*A*X=A^t*B$$

make A quadratic by leftmultiplication with A^t ... with gives this *normalequation*

❷ $\downarrow$

$$(A^t*A)^{-1}*(A^t*A)*X=(A^t*A)^{-1}*A^t*B$$
$$X=\underbrace{(A^t*A)^{-1}*A^t}_{\mathbf{mpi1(A)}}*B$$

if $A^t * A$ invertible, invert it and shorten the normalequation to get the solution X free

Here is our encoding *macro* for the product term in front of B:

Definition.

$$\mathbf{mpi1(A)} \overset{\text{def}}{=} (A^t * A)^{-1} * A^t \tag{1.9}$$

and read it as 'the Moore-Penrose-*Pseudoinverse* of A'.

Remark.

1. We abbreviate the frequently occurring term $(A^t * A)^{-1} * A^t$ with the notation *mpi1(A)*. Occasionally we have to be able to unpack ('decode') this notation into the matrix product on the right-hand side of the formula (1.9).

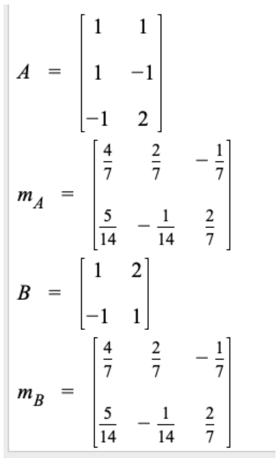
2. The '1' in the identifier $mpi1(A)$ should remind you of the assumed *invertibility* of $A^t * A$ as a necessary condition to be able to form $mpi1$ at all.

3. We can define (1.9) in Eigenmath as an *executable* formula, see sec. 1.1.9:

```
mpi1(A)=dot(inv(dot(transpose(A),A)),transpose(A))
```

Example.
EIGENMATH user input: EIGENMATH output:

```
mpi1(A)=dot(inv(dot(transpose(A),A)),
            transpose(A))

A=((1,1),(1,-1),(-1,2))
A
mA = mpi1(A)
mA

B=((1,2),(-1,1))
B
mB = mpi1(A)
mB
```



$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$m_A = \begin{bmatrix} \frac{4}{7} & \frac{2}{7} & -\frac{1}{7} \\ \frac{5}{14} & -\frac{1}{14} & \frac{2}{7} \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}$$

$$m_B = \begin{bmatrix} \frac{4}{7} & \frac{2}{7} & -\frac{1}{7} \\ \frac{5}{14} & -\frac{1}{14} & \frac{2}{7} \end{bmatrix}$$

Be warned: $mpi1$ is not always successful. Try e.g. the matrix `No=` $\binom{1\,2\,3}{2\,3\,6}$. Observe the answer of EIGENMATH. Therefore our journey into the realm of the world of pseudoinveres is not over with the construction of $mpi1$ ...

1.1.11 Exercise: Projection onto the column space of a vector aka $n\times 1$- matrix

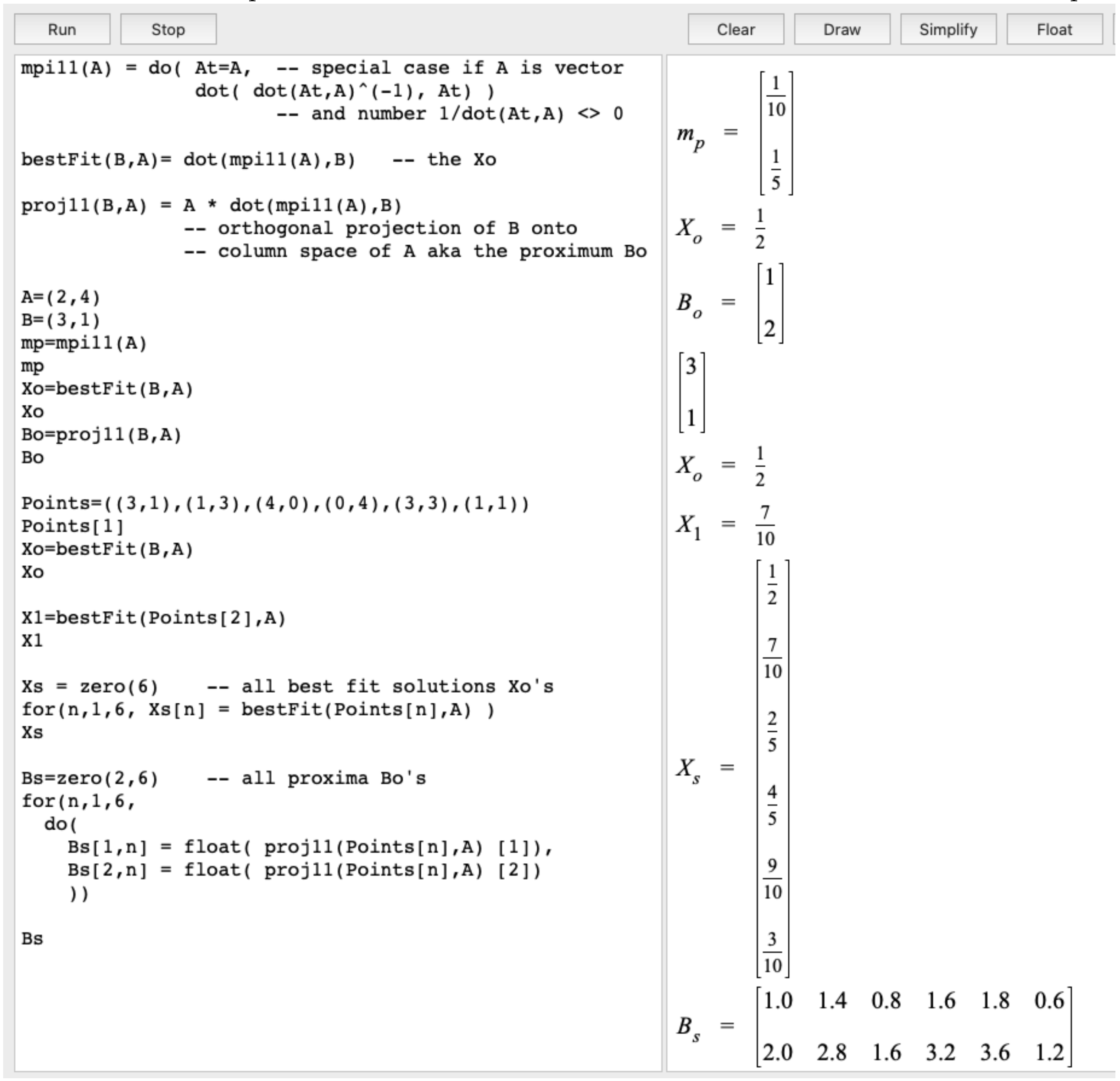
Redo exercise 1.1.8 by splitting the term for `oProj(B,A)` in two phases e.g. by defining

```
mpi11(A) = do( At=A,                          -- only if A is a vector
               dot( dot(At,A)^(-1), At) )     -- and 1/dot(At,A) <> 0

proj11(B,A) = ?                    -- orthogonal projection of B onto  A
```

Use the piontslist `Points=((3,1),(1,3),(4,0),(0,4),(3,3),(1,1))` as test list.

- How should `proj11` then be coded?
- How would the EIGENMATH-definition of the function `mpi11` change, if A is a matrix (not a vector!)?
- Calculate the image points of all points in the pointlist under the action of `proj11` and visualizise all data (pointlist, image list, vector A) in a figure.
- Look at the following solution screenshot not before you had a try to solve it for yourself or you have got a good visual representation in your brain.

EIGENMATH user input: EIGENMATH output:



```
mpi11(A) = do( At=A,  -- special case if A is vector
               dot( dot(At,A)^(-1), At) )
                      -- and number 1/dot(At,A) <> 0

bestFit(B,A)= dot(mpi11(A),B)   -- the Xo

proj11(B,A) = A * dot(mpi11(A),B)
              -- orthogonal projection of B onto
              -- column space of A aka the proximum Bo

A=(2,4)
B=(3,1)
mp=mpi11(A)
mp
Xo=bestFit(B,A)
Xo
Bo=proj11(B,A)
Bo

Points=((3,1),(1,3),(4,0),(0,4),(3,3),(1,1))
Points[1]
Xo=bestFit(B,A)
Xo

X1=bestFit(Points[2],A)
X1

Xs = zero(6)    -- all best fit solutions Xo's
for(n,1,6, Xs[n] = bestFit(Points[n],A) )
Xs

Bs=zero(2,6)    -- all proxima Bo's
for(n,1,6,
  do(
    Bs[1,n] = float( proj11(Points[n],A) [1]),
    Bs[2,n] = float( proj11(Points[n],A) [2])
    ))

Bs
```

$$m_p = \begin{bmatrix} \frac{1}{10} \\ \frac{1}{5} \end{bmatrix}$$

$$X_o = \frac{1}{2}$$

$$B_o = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$X_o = \frac{1}{2}$$

$$X_1 = \frac{7}{10}$$

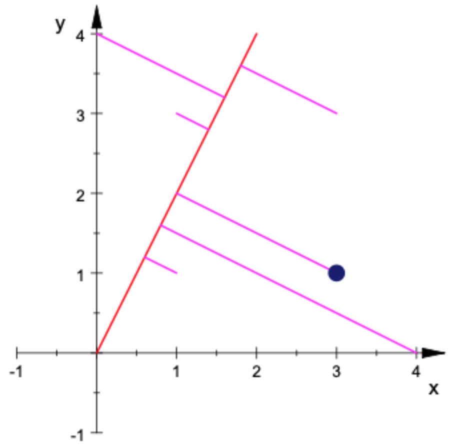
$$X_s = \begin{bmatrix} \frac{1}{2} \\ \frac{7}{10} \\ \frac{2}{5} \\ \frac{4}{5} \\ \frac{9}{10} \\ \frac{3}{10} \end{bmatrix}$$

$$B_s = \begin{bmatrix} 1.0 & 1.4 & 0.8 & 1.6 & 1.8 & 0.6 \\ 2.0 & 2.8 & 1.6 & 3.2 & 3.6 & 1.2 \end{bmatrix}$$

We see, that the MOORE-PENROSE-*Pseudoinverse mpi*11 for a vector A

- allows to calculate the best fit solution Xo given A and B via `dot(mpi11(A),B)`
- allows to calculate the corresponding proximum via `A * dot(mpi11(A),B)`
- is a crucial part of the solution of this approximation problem.

The figure shows the orthogonal projections of the `Points[i]` onto the column space (the red line) of the matrix A. Argue, why this fact explains their *optimality* resp. their *best fitting*.



Exercise. Show, by inspecting the coding in the last EIGENMATH session, that one can alternatively write down the definition for `proj11` without citing the factor `mpi11` in the following way:

```
proj11a(B,A) = do( At=A,  -- special case if A is vector and 1/dot(At,A) <> 0
                   A * dot( dot(At,A)^(-1), At, B))
```

1.1.12 Definition: Projection onto the column space of a matrix

The foregoing section give a good hint how to generalize the definitions above to matrices.

Definition.

$$\mathbf{proj1}(\mathbf{B},\mathbf{A}) \overset{\text{def}}{=} A*(A^t*A)^{-1}*A^t*B = A*mpi1(A)*B \tag{1.10}$$

and read it as the *projection of B onto (the column space of)* of A.

Remark.

1. We can define (1.10) in EIGENMATH as an *executable* formula as follows:

```
proj1(B,A) = do( At=transpose(A),  --if A is matrix with detA<>0
                 dot(A, inv(dot(At,A)), At, B))
```

2. Remember that the term $(A^t * A)^{-1} * A^t$ is identified os the *mpi1(A)*.

3. The '1' in the identifier $proj1(A)$ should remind you of the assumed *invertibility* of $A^t * A$ as a necessary condition to be able to build $proj1$ at all.

1.1.13 Summary of concepts and associated Eigenmath-toolbox

Lexicon: Math vs. Eigenmath

	mathematical notation	Eigenmath notation
5	the linear equations $A * X = B$ in matrix form	`A, X, B` as matrices
6	the Moore-Penrose-*Inverse* $mpi(A) = (A^t * A)^{-1} * A^t$	`mpi(A)`
7	the orthogonal projection of B onto A: $A * mpi(A) * B$	`proj(B,A)`
8	the best fit solution X_o of $AX = B$: $X_o = mpi(A) * B$	`Xo = dot(mpi(A),B)`
9	the proximum Bo to B: dist(P, Q)	`dot(A,inv(dot(At,A)),At,B))` $\therefore$ `proj(B,A))`

The Eigenmath commands of the lexicon before are shown in the following file *mpiBox.txt*.

Eigenmath **toolbox:**

```
---------- BEGIN File mpiBox.txt ---------

mpi(A) = test( rank(A)=1,     --if A vector & dot(A,A)<>0
                A * 1/dot(A,A) ,
               rank(A)>1,     --if A matrix & det(A)<>0
                do(At=transpose(A), dot(inv(dot(At,A)),At)) )

proj(B,A) = test(
              rank(A)=1,      --if A vector & dot(A,A)<>0
               dot(A,B,A) / dot(A,A) ,
              rank(A)>1,      --if A matrix & det(A)<>0
               dot(A, mpi(A), B) )

Xfit(B,A) = dot(mpi(A),B)     -- the best fit solution Xo
Yfit(B,A) = dot(A, mpi(A), B) -- the proximum Bo

---------- END mpiBox.txt ----------------
```

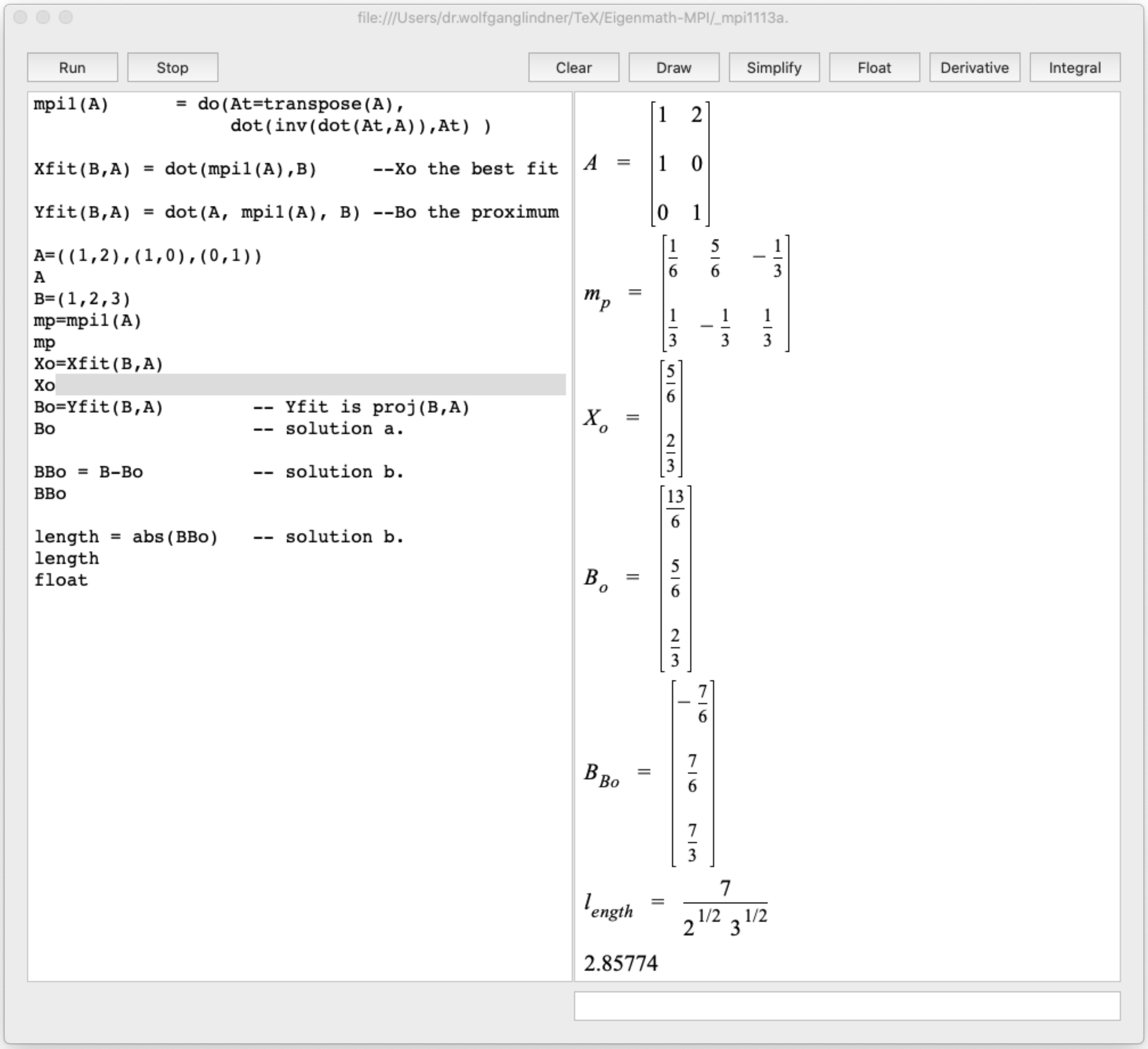
1.1.14 Example: Projection onto a plane

a. Project the point $(1; 2; 3)$ onto the plane with the equation $x - y - 2z = 0$.
b. Calculate the projection vector and his length.
c. What is the column space $Col(A)$?

Solution.
The projection plane $E : x - y - 2z = 0$ is the set of all points (x, y, z) with

$$\begin{array}{rrcl} & (x,y,z) & : & x = y + 2z \\ \therefore & (y+2z, y, z) & : & x, y, z \in \mathbb{R} \\ \therefore & y(1,1,0) + z(2,0,1) & : & y, z \in \mathbb{R} \end{array}$$

that is the plane E is spanned by the vectors $(1, 1, 0)$ and $(2, 0, 1)$. These two (basis) vectors of E form the columns (the 'column space') of a matrix A onto which we project $B = (1; 2; 3)$. Let's do EIGENMATH the work for us!
EIGENMATH user input: EIGENMATH output:



```
mpi1(A)      = do(At=transpose(A),
                  dot(inv(dot(At,A)),At) )

Xfit(B,A) = dot(mpi1(A),B)     --Xo the best fit

Yfit(B,A) = dot(A, mpi1(A), B) --Bo the proximum

A=((1,2),(1,0),(0,1))
A
B=(1,2,3)
mp=mpi1(A)
mp
Xo=Xfit(B,A)
Xo
Bo=Yfit(B,A)        -- Yfit is proj(B,A)
Bo                  -- solution a.

BBo = B-Bo          -- solution b.
BBo

length = abs(BBo)   -- solution b.
length
float
```

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

$$m_p = \begin{bmatrix} \frac{1}{6} & \frac{5}{6} & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

$$X_o = \begin{bmatrix} \frac{5}{6} \\ \frac{2}{3} \end{bmatrix}$$

$$B_o = \begin{bmatrix} \frac{13}{6} \\ \frac{5}{6} \\ \frac{2}{3} \end{bmatrix}$$

$$B_{Bo} = \begin{bmatrix} -\frac{7}{6} \\ \frac{7}{6} \\ \frac{7}{3} \end{bmatrix}$$

$$l_{ength} = \frac{7}{2^{1/2}\, 3^{1/2}}$$

2.85774

We verify our solution by means of a visualization of the whole scene.
Check the results for plausibility using the figure.

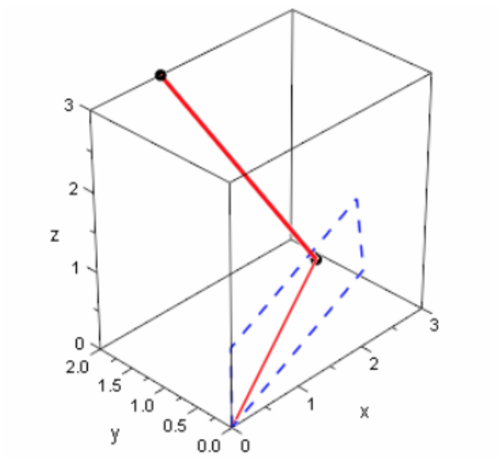
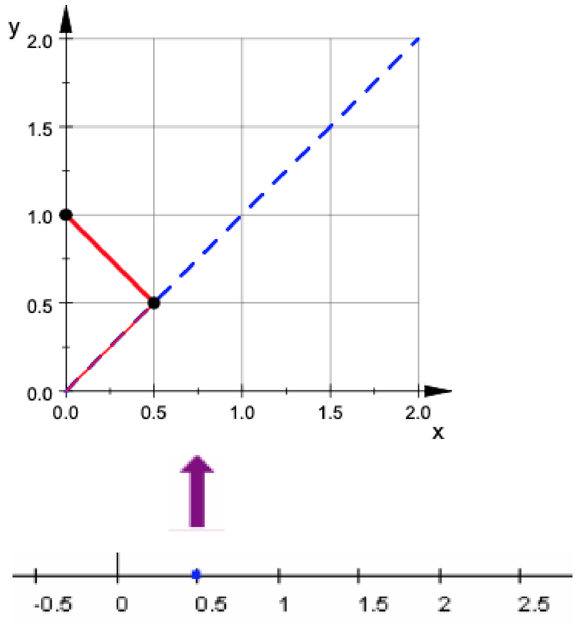


Figure 3: Representation of the projection of B onto the column space $Col(A) = E$; E is shown in parametric form by the blue dashed parallelogram. The edges vectors starting from the zero point are the basis vectors of E (alias the columns of A) of the parallelogram and 'span' the whole $E = Col(A)$.

1.1.15 interpretation of the 'simplest unsolvable LS in the world'

The following picture shows a geometric interpretation of Ex.1.1.1.a as an orthogonal projection of $B = (0; 1)$ onto the column space of the $A = (1; 1)$:



.. leads to the best fit (proximum) `Bo`=(0.5; 0.5) of $A * X = B$ in $\mathbb{R}^2$.

The exact `Xo`=0.5 best fit solution (from normalequation) in $\mathbb{R}$ ..

a. Explain: The proximum $Bo = (1/2; 1/2)$ of the unsolvable LS `{1x=0 & 1x=1}` is the vertical projection of $B = (0, 1)$ onto the straight line through the origin and $A = (1, 1)$. The best fit solution $Xo = 1/2$ is the stretching factor to reach this projection base point.
b. Using the figure, interpret the intuitive solution $x = 1/2$ from Ex.1.1.1.a geometrically.
c. Calculate the best approximation error vector $e = B - Proj(B, A)$ and the amount of this error. Where do you see the error vector e in the image?
Why do we denote e or $|e|$ as a mistake?

1.2 Problems.

1.2.1 Small relaxation exercises for the projection formula.

Calculate the projection of
a. $R = (1, 2)$ on the straight line through the origin through $U = (4, 1)$.
b. $R = (1, 2)$ on the straight line through the origin through $U = (-3, 1)$.
c. $X = (1, 2)$ on the straight line through $P = (0, 5)$ and $Q = (5, 0)$.
Make a sketch with resp. without EIGENMATH.
First estimate the result by eye, then with the coordinate system.

1.2.2 Projection on straight line through origin.

To what extent do Ex.1.1.1.b and Ex.1.1.1.c represent the same problem and
to what extent are 1.3, 1.4 and 1.5 two different solutions to the same problem.
Compare the two solution methods for solving this distance problem.
Do you have an idea for another different solution method?

1.2.3 Distance of a point from a plane.

The distance of a point $R = (R1, R2, R3)$ from the plane $E : X = P + rU + sV$ is the ...... of the vector $F - R$, where F is the ....
The point F can be calculated from three conditions:
(1) $R - F \perp U$ that is $(R - F) \bullet U = 0$
(2) $R - F \perp V$ that is $(R - F) \bullet V = 0$
(3) $F \in E$ that is $F = \ldots$ with $r, s \in R$ suitable.

a. Make a sketch of the situation.
b. Calculate the distance between point R and plane E.
c. specifically for $R = (3, 1, -2)$ and $E : X = (2, 0, 0) + r(1, 1, 0) + s(2, 0, 1)$.
d. for a general R and
e. derive an EIGENMATH formula for the distance $dist(R, E) = |R - F|$ from R to E.

1.2.4 Compromise solution for parallel straight lines.

Solve Ex.1.1.1.d with paper and pencil without EIGENMATH.
Interpret the solution and compare it to the 'intuitive guess'.

1.2.5 Best fit solution for the simplest unsolvable LS in the world.

Solve Ex.1.1.1.a with paper and pencil. Alternatively, solve with EIGENMATH.
Interpret the solution and compare it to your intuition.

1.2.6 Best fit solution for weight gain.

Solve Ex.1.1.1.e with the help of EIGENMATH. Interpret the result.
Compare it to your intuitive guess.

1.2.7 Projection on straight line through origin.

Verify using EIGENMATH, that the orthogonal projection onto the straight line through the origin with equation $y = mx$ has the matrix

$$\begin{pmatrix} \frac{1}{m^2+1} & \frac{m}{m^2+1} \\ \frac{m}{m^2+1} & \frac{m^2}{m^2+1} \end{pmatrix}$$

1.2.8 Thinking big: solutions-in-one-strike.

Solve problems P.1.4.4, P.1.4.5, P.1.4. and Ex.1.2.1 using the concepts *mpi* and *proj* in 'a-one-liner' each.

1.2.9 Distance between two crooked straight lines.

Two straight lines $g : X = P + kU$ and $h : X = Q + mV$ are called *crooked* (in symbols: $g \bowtie h$), if they do not intersect and their direction vectors are not parallel.
Their distance $dist(g \bowtie h)$ is then the ... of that vector $\overrightarrow{SR}$ with $R \in g$ or $S \in h$, which is perpendicular both on U and on V.

The points R and S result from the 4 conditions: (fill in the dots ♡)
(1) $R - S \perp U$ that is ...
(2) $R - S \perp V$ that is ...
(3) $R \in g$ that is $R = \ldots$ with $r \in R$
(4) $S \in h$ that is $S = \ldots$ with $s \in R$ r, s suitable.

a. Make a sketch of the situation.
b. Consider that R and S and thus $dist(g \bowtie h) = |R - S|$ can be calculated from the following 2×2 - LS for (r, s):
† $(P + rU - Q - sV) \bullet U = 0$
‡ $(P + rU - Q - sV) \bullet V = 0$
c. Write this LS in matrix form.
d. Justify that the straight lines $g : X = (1, 2, 0) + k(1, 0, 1)$ and $h : X = (0, 2, 3) + l(0, 1, 0)$ are crooked and calculate their distance.

h. Derive a general Eigenmath formula to calculate the distance of two crooked straight lines.

1.2.10 Proof of concept - automatic proposition proving with Eigenmath.

In the foregoing chapter we tried hard to mimic a mathematical concept in the language of Eigenmath as similar as possible. For example, the original definition of orthogonal projection in 1.3.1 looks very long and complicated and is hard to remember. So George Weigt[14] suggested a shorter term (2), which is cooler to memorize.
The following code snippet send by George shows both definitions of this concept:

```
A = (A1,A2,A3)
B = (B1,B2,B3)

oProj(B,A) = dot(A, dot( dot(A,A)^(-1), A, B))   -- Wolfgangs definition (1)
oProj1(B,A) = dot(A,B,A) / dot(A,A)              -- Georges definition (2)

T1 = oProj(B,A)
T2 = oProj1(B,A)

T1 - T2.                     -- (3)
T1 == T2                     -- (4)
```

a. Run this code snippet in a fresh Eigenmath session.
b. Argue, which heuristic may George have lead to term (2).
Think about the semantic of (1) and (2) and give pros and cons for both terms.
c. George gave to arguments (3) and (4) for the suspected equivalence of the two different terms for the defining formula for orthogonal projection of a general vector B onto a general vector A. Explain.
d. Is definition (2) valid also for matrices (i.e. $rank(A) > 1$) ?

[14]the author and maintainer of Eigenmath

2 Best Fittings

In this chapter we consider applications of the acquired insights and solution techniques for unsolvable linear systems of equations using a few examples from natural, social and engineering sciences. Before that, in 2.1 and 2.2 we repeat two familiar geometrical problems in order to get in tune with the new methods. Subsequently, the single-line state-of-the-art solution of the so-called regression problem of elementary exploratory data analysis is demonstrated as a paradigmatic example of the adjustment calculation.

2.1 EIGENMATH lab: third solution of the distance problem.

What is the distance between the point $R(1,3)$ and the straight line with the equation $y = 0.5x + 1$?

Solution. The previous approaches to Ex.1.1.1.b. managed without the use of matrices. Does a re-interpretation of the problem or the approach with matrices create new insights? If we write g vectorially in parameter form $X = P + t(Q - P)$ with $P = (0,1)$ and $Q = (2,2)$ living on g, the following applies.
The measured distance $|R - P - t(Q - P)|$ should be minimal. Since there is no t such that $R - P - t(Q - P) = 0$ (that would be 'at most minimal'), we look for a compensation solution for the unknown t with $R - P - t(Q - P) \approx 0$ that is

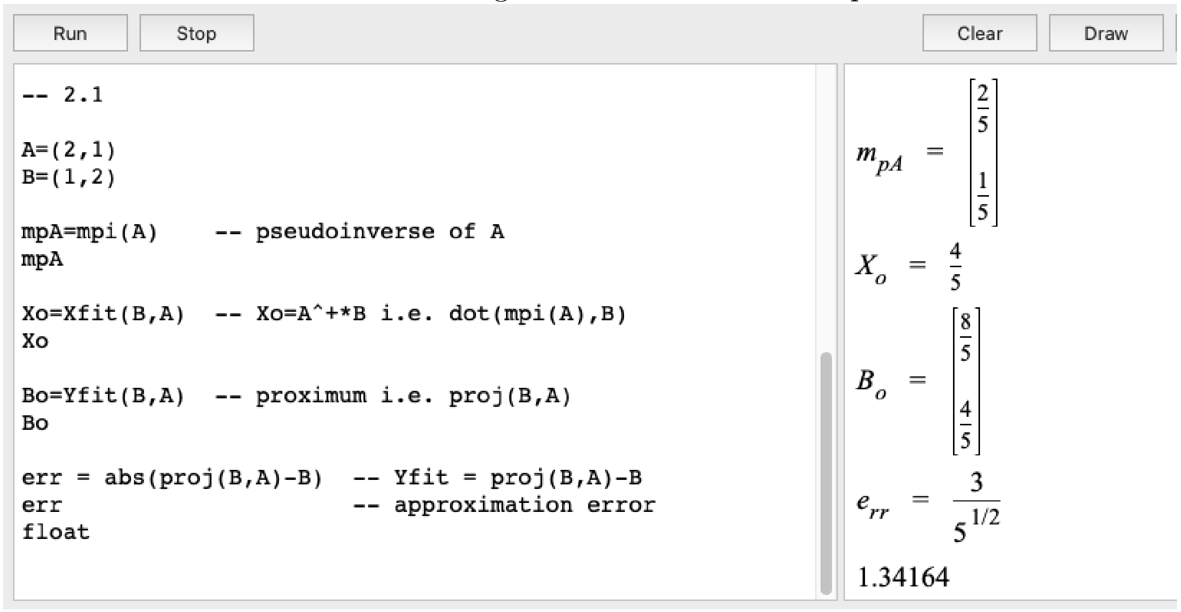
$$\begin{aligned} R - P &\approx t(Q - P) \\ \begin{pmatrix}2\\1\end{pmatrix} t &\approx \begin{pmatrix}1\\2\end{pmatrix} \end{aligned}$$

This over-determined linear equation in matrixform $\binom{2}{1}t \approx \binom{1}{2}$ for the unkown t does not have a solution.

a. semi-automatic solution with *mpi*. First we demonstate a semi-automatic calculation of the best fit in form of triples (EIGENMATH `command`, math form, [result), so you can follow the reasoning by hand, brain and EIGENMATH. We try to solve for the unknown t using multiplications with matrices::

EIGENMATH:	*Math* :	[Gives:
`do( A=(2,1), B=1,2), X=(x))`	$A = \binom{2}{1}, B = \binom{1}{2}, X = (x)$	[
`mpi(A)`	$(A^t * A)^{-1} * A^t$	$[(\frac{2}{5}\ \frac{1}{5})$
`mpi(A)*B`	$(A^t * A)^{-1} * A^t * B$	$[(\frac{4}{5})$
	calculate the proximum Yo:	
`A*mpi(A)*B`	$A * (A^t * A)^{-1} * A^t * B$	$[\binom{8/5}{4/5}$
`proj(B,A)`	$A * (A^t * A)^{-1} * A^t * B$	$[\binom{8/5}{4/5}$
`proj(B,A)-B`	$(A^t * A)^{-1} * A^t * B - B$	$[\binom{3/5}{-6/5})$
`abs( proj(B,A)-B )`	$\lvert(A^t * A)^{-1} * A^t * B - B\rvert$	$[\approx 1.34)$

b. full-automatic solution with *mpi*.
Second we calculate the results using the function from our mpiBox.



```
-- 2.1

A=(2,1)
B=(1,2)

mpA=mpi(A)    -- pseudoinverse of A
mpA

Xo=Xfit(B,A)  -- Xo=A^+*B i.e. dot(mpi(A),B)
Xo

Bo=Yfit(B,A)  -- proximum i.e. proj(B,A)
Bo

err = abs(proj(B,A)-B)   -- Yfit = proj(B,A)-B
err                      -- approximation error
float
```

$$m_{pA} = \begin{bmatrix} \frac{2}{5} \\ \frac{1}{5} \end{bmatrix}$$

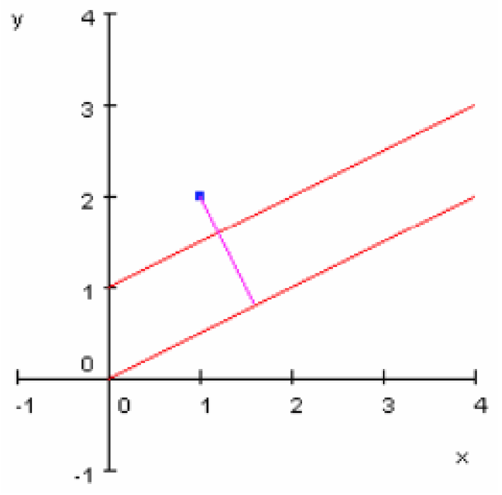
$$X_o = \frac{4}{5}$$

$$B_o = \begin{bmatrix} \frac{8}{5} \\ \frac{4}{5} \end{bmatrix}$$

$$e_{rr} = \frac{3}{5^{1/2}}$$

1.34164

We interpret the results looking at the graphic scene.



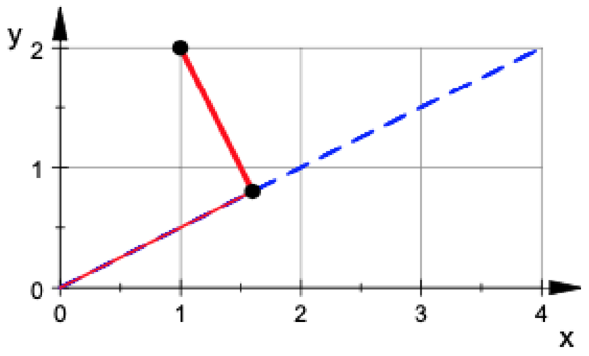


Fig.a: drawing of the projection of B onto the 1-dimensional column space of A. Note: the column space of g is the same, because g is parallel to the column space of A.

Fig.b: the projection of B onto the 1-dimensional column space (straight line through origin) of A.

2.1.1 Exercise: 3D distance as fitting problem.

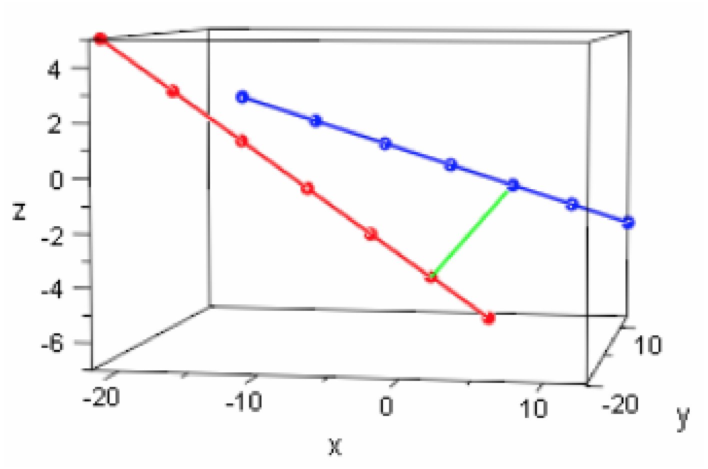
Calculate the distance of point $R(6, -3, 12)$ to the plane $E : X = (7, 2, 5) + r(1, 3, 1) + s(-2, 1, 1)$ analog to the example above. $[Result : 62]$

2.2 Eigenmath Lab: Distance between crooked straight lines.

Calculate the distance between the skewed straight lines $g : X = (1, -1.0) + r(4.6, -1)$ and $h : X = (-10, -1, -1) + s(-4, -5.2)$ by means of a best fit calculation.

Solution.

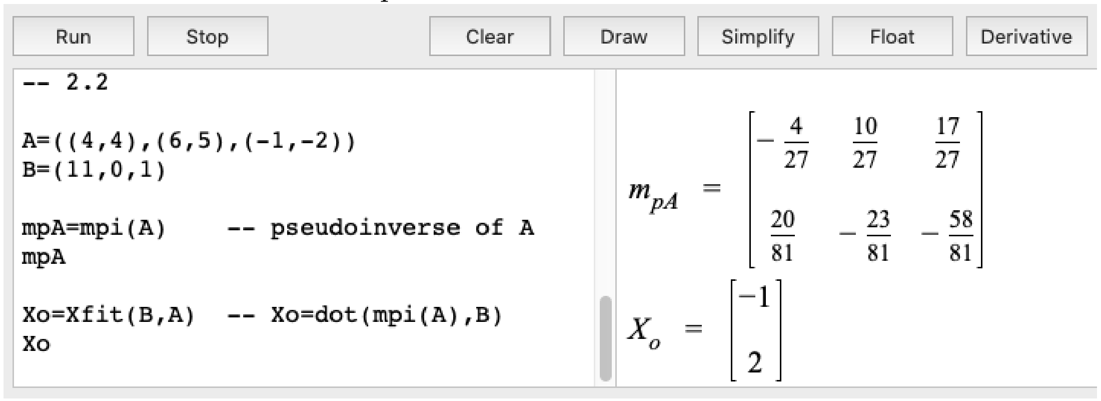
The solution takes place according to the strategy from laboratory 2.1 in three steps.
Step 1: Formulate the distance problem as a minimal problem:
$|((1, -1.0) + r(4.6, -1)) - ((-10, -1, -1) + s(-4, -5.2))|$ is to be minimized!
Step 2: Reformulate the minimal problem as matrix equation $A * X = B$.
Step 3: Solve the system $AX = B$ using *mpi*.



Step **1** ok. Done.
Step **2** We write the formula of step 1 as matrix equation.

$$\begin{bmatrix} 4 & 4 \\ 6 & 5 \\ -1 & -2 \end{bmatrix} \bullet \begin{bmatrix} t \\ s \end{bmatrix} - \begin{bmatrix} -11 \\ 0 \\ -1 \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{i.e.} \quad \begin{bmatrix} 4 & 4 \\ 6 & 5 \\ -1 & -2 \end{bmatrix} \bullet \begin{bmatrix} t \\ s \end{bmatrix} \approx \begin{bmatrix} 11 \\ 0 \\ 1 \end{bmatrix}.$$

Step **3** Solve with Eigenmath mpiBox:



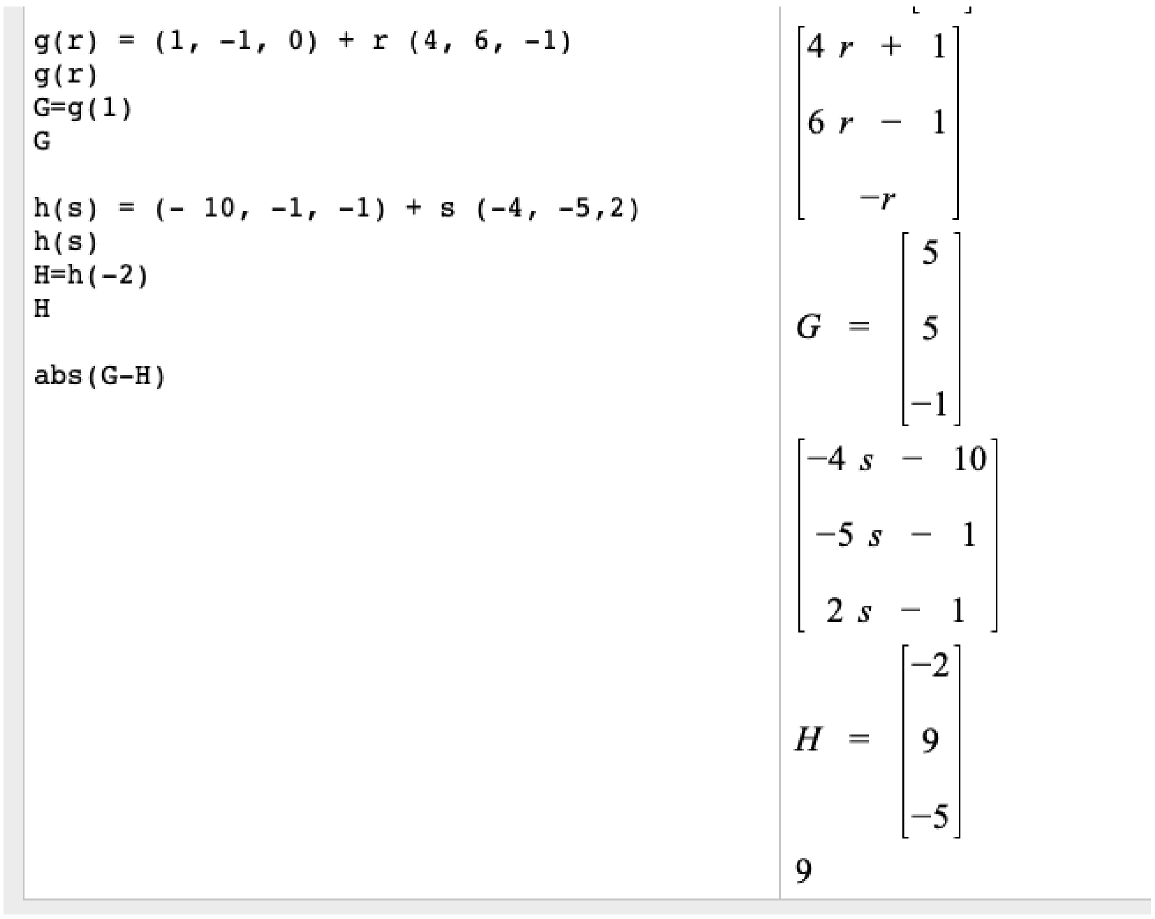
```
-- 2.2

A=((4,4),(6,5),(-1,-2))
B=(11,0,1)

mpA=mpi(A)    -- pseudoinverse of A
mpA

Xo=Xfit(B,A)  -- Xo=dot(mpi(A),B)
Xo
```

$$m_{pA} = \begin{bmatrix} -\frac{4}{27} & \frac{10}{27} & \frac{17}{27} \\ \frac{20}{81} & -\frac{23}{81} & -\frac{58}{81} \end{bmatrix}$$

$$X_o = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

We now calculate those points G on g and H on h where the minimal distance take place. We then check their distance: it should be the same value as the best fit!

EIGENMATH user input: EIGENMATH output:



```
g(r) = (1, -1, 0) + r (4, 6, -1)
g(r)
G=g(1)
G

h(s) = (- 10, -1, -1) + s (-4, -5,2)
h(s)
H=h(-2)
H

abs(G-H)
```

$$\begin{bmatrix} 4r + 1 \\ 6r - 1 \\ -r \end{bmatrix}$$

$$G = \begin{bmatrix} 5 \\ 5 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -4s - 10 \\ -5s - 1 \\ 2s - 1 \end{bmatrix}$$

$$H = \begin{bmatrix} -2 \\ 9 \\ -5 \end{bmatrix}$$

9

- If you are not interested in the points G and H: how could you determine the distance of g and h with only one EIGENMATH command?

2.3 EIGENMATH laboratory: regression line as best fit problem.

We now solve problem Ex.1.1.1e - the weight gain - using matrices.

2.3.1 Solving a scaled-down problem

However, we will first look at a scaled-down prototype version of the problem that will allow us to better oversee the details. The new smaller one is:

Problem: *Find the straight line that runs closest to the three points* $(0;3),(1;0)$ *and* $(2;0)$.

We proceed according to our three-step solution method:
1. Set the up the problem as a minimal problem.
2. Reformulate the minimal problem in matrix form $A*X=B$.
3. Solve this minimal problem in EIGENMATH using *mpi*-pseudoinverse method.

Step **1** Ansatz: the straight line you are looking for has the equation $y=mx+b$.
Point $(0;3)$ lies on $y=m*x+b$, if $m*0+b*1=3$. (1)
Point $(1;0)$ lies on $y=m*x+b$, if $m*1+b*1=0$. (2)
Point $(2;0)$ lies on $y=m*x+b$, if $m*2+b*1=0$. (3)

This overdetermined 3x2 LS for the unknown pair $(m;n)$ has *no* solution.

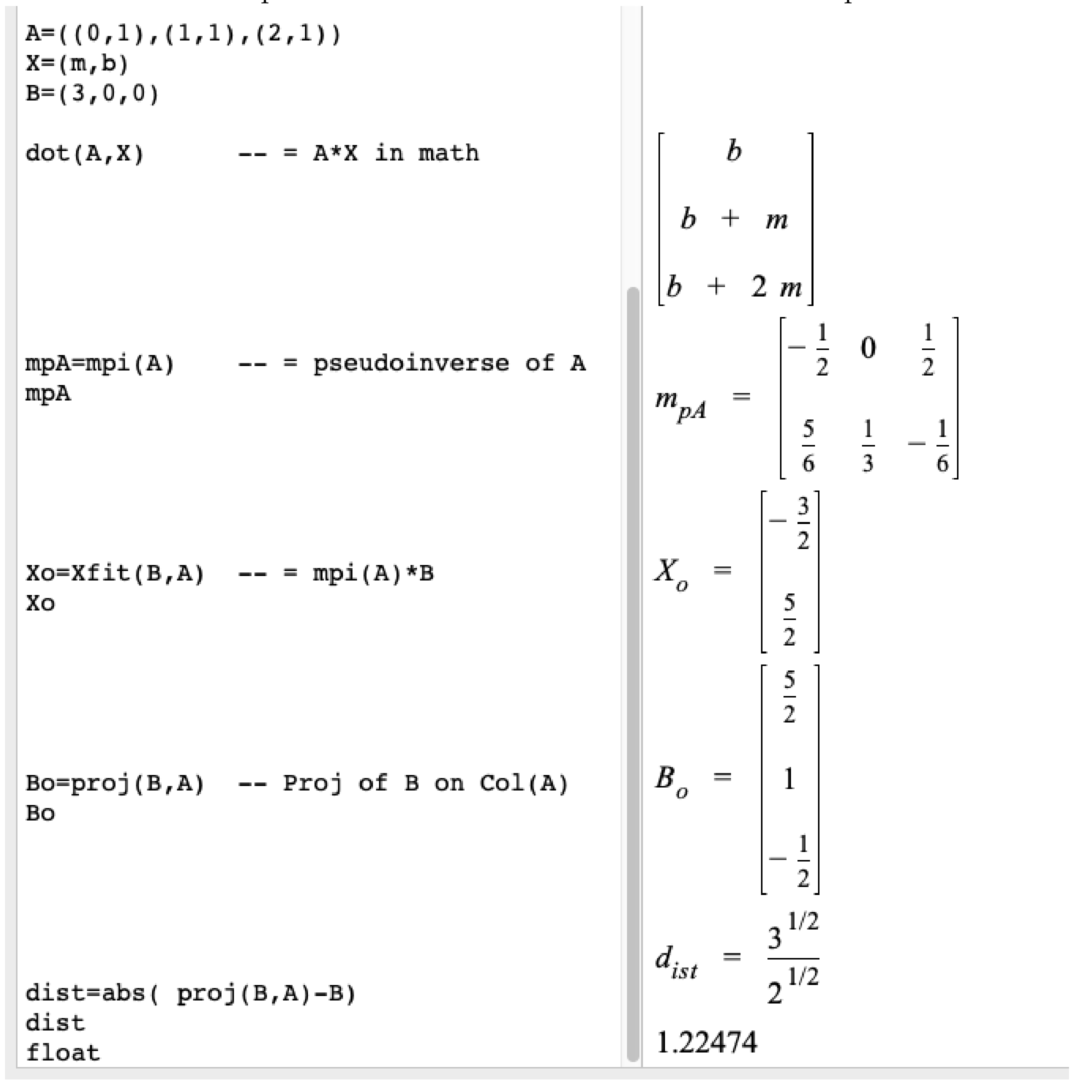
Step **2** Minimal problem in matrix form: the 3 equations (1), (2), (3) are put as 3 lines in 2 matrices, named A for the LHS and B for the RHS, therefore leading to the matrix equation

$$\begin{array}{ccccc} A & & X & & B \\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} & \cdot & \begin{bmatrix} m \\ b \end{bmatrix} & \approx & \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} \end{array}$$

Step **3** Each $A*X$ is a linear combination of the columns of A, i.e. $A*X$ lies in the plane $Col(A)$ formed by the columns $(1,1,1)$ and $(0,1,2)$ of the matrix, the well known column space of A. In this plane we look for $B_o \in Col(A)$, which is the nearest point to $B \notin Col(A)$:[15] this point is known to be the base point B_o the orthogonal projection of B onto the column space of A - and this foot point is produced by the pseudo inverse*mpi*, as we have often observed. Therefore:

[15]Remember: the point B i.e. the RHS of LS $A*X=B$, is not included in $Col(A)$, because the system is not solvable!

EIGENMATH user input: EIGENMATH output:



```
A=((0,1),(1,1),(2,1))
X=(m,b)
B=(3,0,0)

dot(A,X)       -- = A*X in math

mpA=mpi(A)     -- = pseudoinverse of A
mpA

Xo=Xfit(B,A)   -- = mpi(A)*B
Xo

Bo=proj(B,A)   -- Proj of B on Col(A)
Bo

dist=abs( proj(B,A)-B)
dist
float
```

$$\begin{bmatrix} b \\ b + m \\ b + 2m \end{bmatrix}$$

$$m_{pA} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{5}{6} & \frac{1}{3} & -\frac{1}{6} \end{bmatrix}$$

$$X_o = \begin{bmatrix} -\frac{3}{2} \\ \frac{5}{2} \end{bmatrix}$$

$$B_o = \begin{bmatrix} \frac{5}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix}$$

$$d_{ist} = \frac{3^{1/2}}{2^{1/2}}$$

1.22474

... that's it: we get the solution $X_o = ([m = -1.5, b = 2.5)$.

2.3.2 Interpretation.

Inspect and complete the following figures Fig.a and Fig.b with regard to the visibility of the results from 2.3.1.

1. Where does one 'see' the best fit solution vector $(m = -1.5, b = 2.5)$?
2. Calculate the error (vector) *err*.
   Where do you 'see' this error vector in Fig. a, where in Fig. b?
3. Can you *see* these projections in the figures?
   Where can the result or its components be *seen* in Figure a and where in Figure b?

4. Write the proximum $B_o = Proj(B, A)$ as a linear combination of the columns of A. Do not calculate the result. All *necessary* data for this representation is already at hand ..

5. Now use EIGENMATH to calculate d)..

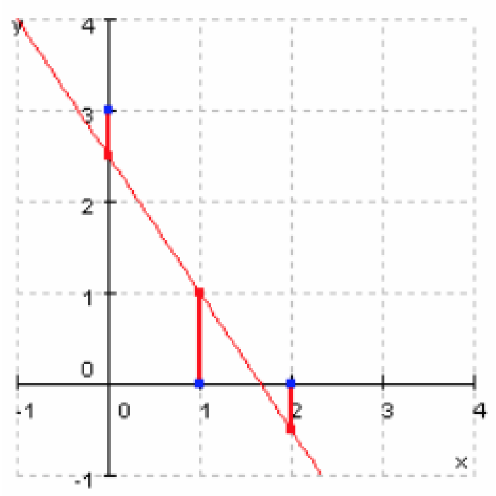


Fig. a: Best fit line through the three data points with localizable coordinates of the error vector err.

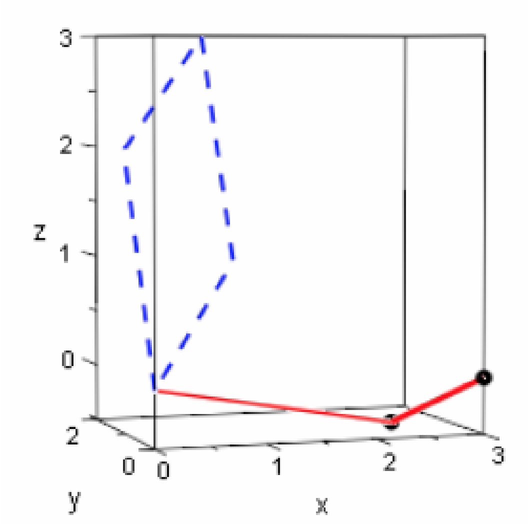


Fig. b: Projection of B onto the 2-dimensional column space (plane) of A. 3D scene rotated by hand control from d.

2.4 ♡EIGENMATH lab: regression line from classic viewpoint.

The following excursus shows interested reader a classic approach to regression lines without pseudoinverse. Nevertheless we will use EIGENMATH to do some tidy computations for us. It could therefore marked with an heart ♡.

Again, we want to solve problem Ex.1.1.1.e, but consider only the first 3 values in the table for simplicity

Woche	3	4	9
kg	4.31	4.51	5.63

The solution is based on the working hypothesis, that all measuring points $X = (x, y)$ are (should) lie approximately on a straight line $g : y = mx + c$. This straight line g - that is, its slope m and its axis intercept c - is to be

Step 1: Standard approach as minimal problem (according to K F GAUSS):
If a point $X = (x, y)$ lies on g, then $y - mx - c = 0$,
if $X = (x, y)$ is not on g, then $y - mx - c \neq 0$.
We therefore start with the ansatz:

$$f(m, c) := (y1 - mx1 - c)^2 + (y1 - mx1 - c)^2 + (y1 - mx1 - c)^2 \qquad \dagger$$

and find m and c such that f (m, c) becomes as small as possible.

Step 2: vectorial formulation of the problem

$$\begin{aligned} f(m, c) &= (y1 - mx1 - c)^2 + (y1 - mx1 - c)^2 + (y1 - mx1 - c)^2 \\ &= (Y - mX - cE)^2 \qquad Y = (y1, y2, y3), X = (x1, x2, x3), E = (1, 1, 1) \\ &= |Y - mX - cE| \\ &= \text{square of the distance from } Y \text{ to } X \in \text{H} \\ & \text{in the auxiliary plane } H : Z = O + mX + cE \end{aligned}$$

2.4.1 ♡♡ mathematical derivation of the classic regression formula.

We conclude from step 2:

$$\begin{aligned} &|Y\text{-}mX\text{-}cE|^2 \text{ minimal} \\ &\Leftrightarrow \quad Y\text{-}mX\text{-}cE \perp H \qquad \text{weil} \ldots\ldots\ldots\ldots\ldots\ldots \\ &\quad (1) \quad (Y\text{-}mX\text{-}cE) \bullet X = 0 \qquad | \bullet E2 \\ &\quad (2) \quad (Y\text{-}mX\text{-}cE) \bullet I = 0 \qquad | \bullet (-E * X) \\ &\quad \Rightarrow (X \bullet Y) * E^2 - (E \bullet X) \cdot (E \bullet Y) + m((E \bullet X)^2 - X^2 E^2) = 0 \quad | (1)+(2) \end{aligned}$$

$$\Leftrightarrow \quad m = \frac{(X \bullet Y) \cdot E^2 - (E \bullet X) \cdot (E \bullet Y)}{X^2 \cdot E^2 - (E \bullet X)^2} \qquad (*m)$$

$$\Leftrightarrow \quad m = \frac{(\sum_{i=1}^{3} x_i y_i).3 - \sum_{i=1}^{3} x_i \cdot \sum_{i=1}^{3} y_i}{(\sum_{i=1}^{3} x_i^2).3 - (\sum_{i=1}^{3} x_i)^2}$$

$$\Leftrightarrow \quad m = \frac{\frac{1}{3}\sum_{i=1}^{3} x_i y_i - \bar{x}.\bar{y}}{\frac{1}{3}(\sum_{i=1}^{3} x_i^2) - \bar{x}^2} \quad \text{mit} \quad \bar{x} := \frac{1}{3}\sum_{i=1}^{3} x_i \ \text{ und } \bar{y} := \frac{1}{3}\sum_{i=1}^{3} y_i \qquad (**m)$$

Therefore we have a formula for m.
Now we get the formula for the unknown c from (2). We have

$$(Y\text{-}mX\text{-}cE) \bullet E = 0 \quad \Leftrightarrow \quad \sum_{i=1}^{3} y_i - m.\sum_{i=1}^{3} x_i - c \cdot 3 = 0$$

$$\Leftrightarrow \quad \bar{y} - m\bar{x} - c = 0 \qquad (*c)$$

Therefore we get the condition $\overline{y} - m\overline{x} - c = 0$ for a point $(\overline{x}, \overline{y})$ to be on the regression line g. Using (*c) in the form $\overline{y} - m\overline{x} = c$ we have the unknown c. So we have g.

2.4.2 Example.

Calculate the equation of the best-fit line through the above 3 pairs of the above formulas ($*m$) and ($*c$) with the calculator to get a feeling for the solution process.
What changes in the derivation or calculation of the regression line g is necessary if you want to have the best fit for all original 7 pairs of values?
We now solve the original 'big data' 1.1.1e problem 'weight gain' using Eigenmath.

Woche	3	4	9	13	18	22	27
kg	4.31	4.51	5.63	6.15	7.26	8.08	8.84

The *solution* consists in the definition of suitable Eigenmath auxiliary functions resp. steps which are driven from the theoretical derivation of 2.4.1.

Eigenmath user input:

```
X = (3,4,9,13,18,22,27)
Y = (4.31,4.51,5.63,6.15,7.26,8.08,8.84)
A = (X,Y)

one1=zero(dim(X))
for(i,1,dim(X), one1[i]=1)
one1

-- compile formula (*m) in 2 phases
m1=dot(X,Y)*one1
m1
m2=dot(X,Y)*dot(one1,one1)
m2

m=( dot(X,Y)*dot(one1,one1) - dot(one1,X)*dot(one1,Y) )/
( dot(X,X)*dot(one1,one1) - dot(one1,X)*dot(one1,X) )
m

mean(X)=sum(i,1,dim(X), X[i])/dim(X)
mean(X)

c=mean(Y)-m*mean(X)
c

y= m*x-c
y
```

Eigenmath output:

$$m_1 = \begin{bmatrix} 708.71 \\ 708.71 \\ 708.71 \\ 708.71 \\ 708.71 \\ 708.71 \\ 708.71 \end{bmatrix}$$

$m_2 = 4960.97$

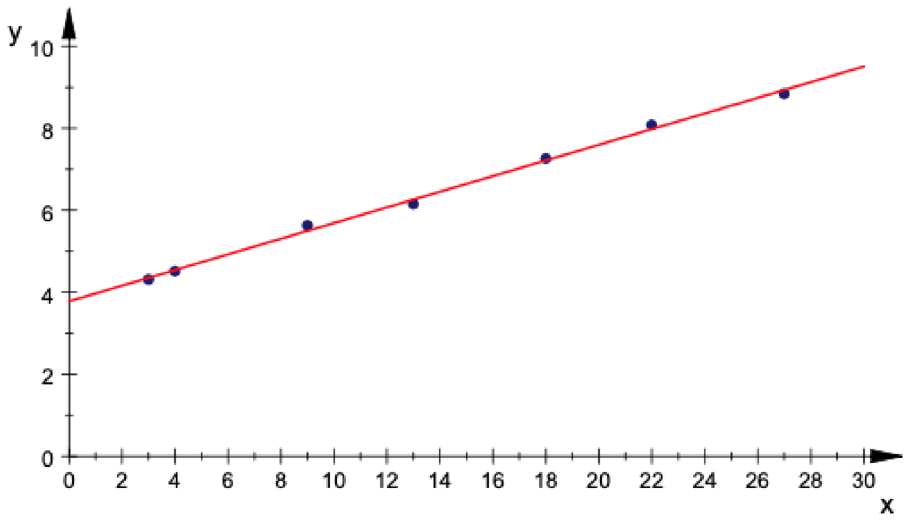
$m = 0.190914$

$\frac{96}{7}$

$c = 3.77889$

$y = 0.190914\,x - 3.77889$

Here is a figure that shows the datapoints and the calculated regression line g.



Remark. Here are some comments about the EIGENMATH code lines.
Vectors X and Y load the data points of the measurements. We combine the two separate data records into a matrix A. To implement the formula $(*m)$ from 2.4.1, we proceed step-by-step by first writing and testing a vector of length A consisting of ones. We code formula $(^*m)$ for the slope m of the straight line via two simpler, separately testable intermediate steps $m1$ and $m2$. To calculate the axis intercept c with formula $(^*c)$, we need a mean value function. We build it with the help of the bulid-in `sum()` function of EIGENMATH .

Code enhancement.
a. Shorten the definition of m in the EIGENMATH session by using $m1$ and $m2$.
b. Create a EIGENMATH function that automatically calculates the regression line g for two data sets X and Y of equal length.

2.4.3 Exercise: The normalequation (NS) and the regression line.

a) Set up the 'normal equation system' (NS) $A^t * A * U = A^t * B$ for the regression line $g : y = mx + c$ for 2.4.1.
Hint: According to 2.4.2 the normal equation system (NS) for the regression line is

$$\mathbf{A^t{*}A{*}U=A^t{*}B} \quad \Leftrightarrow \quad \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix} \bullet \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ & \cdots \\ 1 & x_n \end{bmatrix} \bullet \begin{bmatrix} c \\ m \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix} \bullet \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$$

$$\Leftrightarrow \quad \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \cdot \begin{bmatrix} c \\ m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i \cdot y_i \end{bmatrix}$$

b) Use a. to derive the following classic explicit solution formula for calculating a regression line, which can be found in many statistical textbooks:[16]

$$c := \frac{\left(\sum_{i=1}^{n} x_i^2\right) \cdot \sum_{i=1}^{n} y_i - \left(\sum_{i=1}^{n} x_i\right) \cdot \sum_{i=1}^{n} x_i \cdot y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}, \quad m := \frac{n \cdot \sum_{i=1}^{n} x_i \cdot y_i - \left(\sum_{i=1}^{n} x_i\right) \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

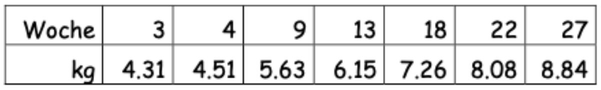
Hint: The solution in follows e.g. according to the well-known CRAMER rule.

[16]The formuli are set in the typical CAS `Derive` font. This CAS was in use in the eighties and nineties. There is an active user group with repository at http://www.austromath.at/dug/ where one can find plenty of interesting material and code snippets.

c) Code the solution formulas for (c, m) from b. using suitable EIGENMATH auxiliary functions or procedure. Now solve exercise Ex.1.1.1e again by calling the corresponding EIGENMATH procedure.

2.5 EIGENMATH lab: State-of-art solution of Regression problem.

These classic, explicit solution formula can hardly be remembered mentally. Here, too, the compact matrix notation shows its brain-relieving and understanding-enhancing effect.
To convince the reader we solve the original 'big data' problem 1.1.1e 'weight gain' again using our EIGENMATH toolbox *mpiBox.txt*. ...



Woche	3	4	9	13	18	22	27
kg	4.31	4.51	5.63	6.15	7.26	8.08	8.84

The *solution* consists in the application of the ready made EIGENMATH toolbox functions. We proceed in two steps only.

TASK		
Given:	n points (x_i, y_i)	$i = 1, 2, \dots n$
Minimize:	$\sum_{i=1}^{n}(mx_i + c - y_i)^2$	*w.r.t.* (c,m)

Step **1** Reformulate the problem using matrix language.
Set up[17]

$$U := \begin{bmatrix} c \\ m \end{bmatrix} \quad B := \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} \quad A := \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ & \cdots \\ 1 & x_n \end{bmatrix}$$

to get the following compact matrix formulation of the regression problem:

$$\sum_{i=1}^{n}(mx_i + c - y_i)^2 = |A * U - B|^2$$

[17]Here U is used as an identifier for the *un*known instead of the familiar X, because in this context we use X for the x-values of the body mass measurements.

Step **2** Solve matrix problem using *mpi*:

EIGENMATH user input:

```
X = (3,4,9,13,18,22,27)
Y = (4.31,4.51,5.63,6.15,7.26,8.08,8.84)
(X,Y) -- the data set
one1=zero(dim(X))
for(i,1,dim(X), one1[i]=1)
-- 1.  set up matrix equation
one1=zero(dim(X))
for(i,1,dim(X), one1[i]=1)
A= transpose((one1,X))
A
U=transpose((c,m))
B=Y
-- 2.  solve problem for U
Uo=dot(mpi(A),B)
Uo
```

EIGENMATH output:

$$\begin{bmatrix} 3 & 4 & 9 & 13 & 18 & 22 & 27 \\ 4.31 & 4.51 & 5.63 & 6.15 & 7.26 & 8.08 & 8.84 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 3 \\ 1 & 4 \\ 1 & 9 \\ 1 & 13 \\ 1 & 18 \\ 1 & 22 \\ 1 & 27 \end{bmatrix}$$

$$U = \begin{bmatrix} c \\ m \end{bmatrix}$$

$$U_o = \begin{bmatrix} 3.77889 \\ 0.190914 \end{bmatrix}$$

Result: we have $U_o[1] = c = 3.77$ and $U_o[2] = m = 0.19$, therefore the regression line has equation $g : y = 0.19x + 3.77$.

2.6 Summary - Solution of linear model problems.

All concrete problems considered so far led, after reformulation with matrices, to one of the following abstract model problems:

model problem described by	Look for solution X of	solution is
regular LS	the linear equation $A * X = B$	$X = A^{-1} * B$ with $det(A) \neq 0$
general LS	the system $A * X = B$	?
linea**r** **best fit**	$A * X \approx B$ or $min_X \lvert A * X - B \rvert$	$X = A * mpi(A) * B$ i.e. $X = A^{+} * B$

Exercise. In the previous summary check resp. add the conditions in the last column. E.g. in cell (4,3) there are 3 conditions: 1. typ$(A) = n \times m$ 2. $n \geq m$ 3. rang$(A) = m$.

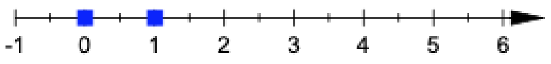
With that summary we could end and close our elementary introduction on pseudoinverse and linear best fit problems. But there is a bit more to say ...

♡ *Mathematics is not formulas, or computations, or even proof, but IDEAS.*♡
Gilbert STRANG, MIT/USA

2.7 Problems.

2.7.1 Arithmetic mean as a solution to a compensation problem.

Express the arithmetic mean of the numbers $1, 2, 3, 4, 5$ as the solution to a linear bet fit problem.
Establish the system of normal equations.
Show a cool solution with the pseudoinverse*mpi*.

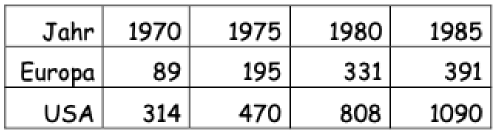


2.7.2 Regression line.

Four points $(1, 1), (2, 4), (3, 2)$ and $(5, 5)$ lie near an unknown straight line.
a. Determine this best approximation line.
b. Formulate the task as a balance problem and solve it in different ways.
c. Compare the procedures. Create a drawing with your hand on paper.

2.7.3 Education budget.

The following table shows the estimated expenditure for the education sector in the USA and Europe in *Eur* from 1970 to 1985 (Unesco Statistical Handbook, 1987, according to [22, p. 377]):

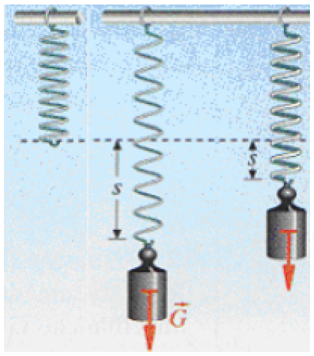


Jahr	1970	1975	1980	1985
Europa	89	195	331	391
USA	314	470	808	1090

a. Forecast estimated values for 1990 and 2000 and compare the estimates with the data collected later (Internet research).
b. Calculate the best-fit line for both data sets 'in one go', i.e. with a LS. Note that both data sets have the same coefficient matrix. (This often occurs in practice.)
c. Make a drawing.

2.7.4 HOOKE's law.

Various weights are attached to a steel spring and the associated extension of the spring is measured. The extension s and the amount of the weight G are recorded in a series of measurements. A measurement resulted in the following values:



G in N	2	3	5	8	
s in cm	7.4	8.8	11.9	16.4	

Calculate the regression line for the data set and use it to predict the measured value for a weighing piece with $G = 10N$. Make a drawing with your favorite grapher app ... .

2.7.5 Best fit parabola.

Calculate the best fit parabola for the data points $(1; 7), (2; 2); (3; 1); (5; 9)$.

2.7.6 HUBBLE-Constant.

Edwin P. HUBBLE (1889-1953) established in the late twenties that the majority of galaxies move at a velocity v proportional to their distance x, i.e. $v = Hx$, away from us. The following table shows the distances in millions of light years for five spiral nebulae and their escape speed in km per sec:

x	500	1400	2100	2900	3000
v	9000	22000	39000	51000	49000

Calculate an approximation for H using a best-fit straight line $v = a + bx$ which is placed trough the above data set.
Remark: Although $a \neq 0$, one can regard b as an approximation for H. The exact calculation of the HUBBLE constant is a current topic in astronomy, as it allows conclusions to be drawn about the age of the universe.

2.7.7 Correlation coefficient.

x	1	2	3	4	5	6	7	8	9	10
y	3	4,1	5	7,2	9,5	10,8	12	13,8	15,5	16,6

a. Determine the regression line for the above measurement table.
b. Find a second regression line by considering the y values as independent values and the x values as an associated dependent values.
c. If $m1$ is the slope of the 1st regression line and $m2$ is the Is the slope of the 2nd regression line, then the number $\rho = \sqrt{m1 \cdot m2}$ is called the *correlation coefficient* of the data series. Calculate the correlation coefficient ρ for the data series.
d. Calculate the correlation coefficient for the HUBBLE constant.

2.7.8 Comparison of methods.

The regression line can also be obtained from the following numerically more stable (cf. e.g. 11, p. 601) approach:

$$(1) \qquad \bar{x} := \sum_{i=1}^{n} x_i \quad \text{und} \quad \bar{\mathrm{y}} := \sum_{i=1}^{n} y_i$$

$$(2) \qquad y = \bar{y} + m(x - \bar{x})$$

$$(3) \qquad m = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})}$$

a. Justify the approach and use it to define a EIGENMATH function RG(X, Y), which returns the solution (c, m).
b. With this approach solve again Ex.1.1.1e.
c. Compare all previously studied methods for calculating a regression line. Vote.

3 Solution Sets of Linear Systems - a view from *mpi*

To calculate inverse matrices of regular LS as well as to determine the solution set of under-determined LS the `Gauss-Jordan` algorithm was the crucial aid. In this chapter we use this trace to construct pseudo-inverses of a matrix and try to develop a solution theory also for *over*-determined LS. Surprisingly, a solution theory for general (also solvable or under-determined) LS is gained: if the system matrix A of a general LS $A * X = B$ is i*nverted 'as far as possible'* we get 1st criteria for the solvability and 2nd a 'general formula', which explicitly specifies the solution set. As a bypass, we get a generalizied version of the `Moore-Penrose` inverse `mpi`.

3.1 The discovery of pseudoinverses

We go into the following EIGENMATH sessions.

3.1.1 Case 1: the Gauss-Jordan algorithm for *regular* LS

Let us study in detail the solution process of the linear 2×2 system[18]

$$\begin{aligned} 1x + 2y &= 3 \\ 2x + 3y &= 6 \end{aligned}$$

This system has $A = \left(\begin{smallmatrix} 1 & 2 \\ 2 & 3 \end{smallmatrix}\right)$ as LHS matrix and $B = \left(\begin{smallmatrix} 3 \\ 6 \end{smallmatrix}\right)$ as RHS. We set up the so called *augmented system matrix* $(A|B) = \left(\begin{smallmatrix} 1 & 2 & 3 \\ 2 & 3 & 6 \end{smallmatrix}\right)$ and track the solution process using so called *elementary* matrix Em.[19] Each Em does exactly one step in reaching the full unity matrix $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ at the LHS, while building the solution vector $\left(\begin{smallmatrix} \cdot \\ \cdot \end{smallmatrix}\right)$ on the RHS. This solution process is called the '*row reduced echelon form*', in short the `RREF`.

Think in the following example by left-multiplying the system $(A|B)$ with the elementary matrix

$$Em(-2, 1, 2)$$

at the mental operation (in your head) as the command

'do -2 times row number 1 of $(A|B)$ and add this to row number 2 of $(A|B)$'.

This is best demonstrated by means of example. Wandering along we make some experiments and observations.

We let do EIGENMATH the dull work for us.

[18]See e.g. [11] for a view on this algorithm from different perspectives.

[19]or elementary left-**m**ultiplication. If we have e.g. in our 2-dimensional case $Em(3, 1, 2) = \left(\begin{smallmatrix} 1 & 0 \\ 3 & 1 \end{smallmatrix}\right)$, then we get $Em(3, 1, 2) * (A|B) = \left(\begin{smallmatrix} 1 & 0 \\ 3 & 1 \end{smallmatrix}\right) * \left(\begin{smallmatrix} 1 & 2 & 3 \\ 2 & 3 & 6 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 2 & 3 \\ 5 & 9 & 15 \end{smallmatrix}\right)$. Be careful: $Em(3, 1, 2)$ *adds* 3 times of row1 to row2 - but it *substitutes* the element at position $[2, 1]$ of the unit matrix with the value 3.

Eigenmath user input: Eigenmath output:

```
n=2
Em(k,i,j) = do( M=unit(n,n), M[j,i]=k, M) -- (em)
A= ((1,2),(2,3)) -- LHS of linear system (LS)
B= (3,6) -- RHS of LS
LS = ((1,2,3),(2,3,6)) -- LS=(A|B)
LS

-- step by step RREF (Row Reduced Echelon Form):

Em(-2,1,2) -- add -2 times row1 to row2
dot( Em(-2,1,2), LS) -- the intermediate result
dot( Em(-1,2,2), Em(-2,1,2), LS)
dot( Em(-2,2,1), Em(-1,2,2), Em(-2,1,2), LS)

RREF = dot( Em(-2,2,1), Em(-1,2,2), Em(-2,1,2))
RREF -- (1)
```

$$L_S = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 0 \end{bmatrix}$$

$$R_{REF} = \begin{bmatrix} -3 & 2 \\ 2 & -1 \end{bmatrix}$$

Comment. The first line gives the definition of $Em(k,i,j)$. We see in the output e.g. $Em(-2,1,2) = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}$. The multiplication with the Em's are shown with intermediate steps:

Start: `Em(-2,1,2)` `Em(-1,2,2)` `Em(-2,1,2)` Finish

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

We read off the unique solution $\binom{3}{0}$ (i.e. choose $x = 3$ and $y = 0$ to fullfill both equations from above) in the last column of the final state - seeing the unity matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ just before it. In code line (1) we save the product of the 3 $Em(..)$ factors in the variable RREF which is $\begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix}$

a. Verify using head or Eigenmath that $RREF * A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, therefore $RREF = A^{-1}$.
b. Show with Eigenmath that $RREF * (A|B)$ gives the solution $\binom{3}{0}$.

3.1.2 Case 2: a singular linear system.

In this case study we change the LHS of the LS above in such a way, that the rows are dependent, e.g. $A = \begin{pmatrix} 1 & -2 \\ 2 & -4 \end{pmatrix}$. Then we have

Eigenmath user input: Eigenmath output:

```
n=2
Em(k,i,j) = do( M=unit(n,n), M[j,i]=k, M)
A= ((1,-2),(2,-4))
B= (3,6)
LS = ((1,-2,3),(2,-4,6)) -- augmented system
(A|B)
LS

dot( Em(-2,1,2), LS) -- (0); building the RREF

RREF = Em(-2,1,2)
RREF                          -- (1)

-- construction a pseudoinverse via RREF

AE = ((1,-2,1,0),(2,-4,0,1)) -- (2)
AE
dot(RREF,AE)                        -- (3)

Apsi = RREF                  -- (4)
Apsi
dot(Apsi,A)                   -- (5)
dot(A,Apsi,A) == A           -- (6)
```

$$L_S = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -4 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

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

$$R_{REF} = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

$$A_E = \begin{bmatrix} 1 & -2 & 1 & 0 \\ 2 & -4 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}$$

$$A_\psi = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 \\ 0 & 0 \end{bmatrix}$$

1

Here the RREF process, which is trying to build the unity matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ at the place of $A = \begin{pmatrix} 1 & -2 \\ 2 & -4 \end{pmatrix}$ inside the augmented matrix $(A|B)$ stops, because there pops up a zero row, see (0). But watch: the RREF process has *constructed the first column of the uinity matrix*, i.e. RREF has **produced partially the unity matrix!**
In 3.1.1 the RREF process produced the inverse matrix RREF= A^{-1} of A (see 3.1.1 (1)). Here the RREF should have produced a similar matrix result in (1), we call it the *partial or pseudo* inverse of A (see (4)), denoted A^-.[20] We do the analogical tests of being a 'pseudo'

[20]Often also called the *generalized* inverse or *g-inverse* (g = generalized) of A ; the notation A^- only use the minus sign $(...)^-$ from A^{-1} to remind on the defect.

inverse in (5) as one would do for the test of being an inverse:

$$A^- * A = A_\psi * A = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix} * \begin{pmatrix} 1 & -2 \\ 2 & -4 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & -2 \\ \mathbf{0} & 1 \end{pmatrix}$$

and test in (6), if $A * A^- * A = A$. The answer of EIGENMATH is 'yes=1'.

Summary: Although A *cannot be inverted* because of $\det(A) = 0$ and therefore *there exists no inverse matrix* 'A^{-1}', the GAUSS-JORDAN algorithm in shape of RREF delivers from the approach for constructing the inverse in (3) a result: the matrix $\mathbf{A}^-$ from (4), which we read from (3) instead of the non-existent inverse A^{-1}.

3.2 Definition: Pseudoinverse.

An $n \times m$ matrix V, which satisfies the equation $A * V * A = A$ for a given $m \times n$ matrix A, is called *a pseudo-inverse of* A. In mathematics, instead of V, one usually writes $\mathbf{A}^-$.

3.2.1 Exercise: Richness of Pseudoinverses.

Fact: Any $m \times n$ matrix (especially any vector!) has a pseudoinverse.[21]

a. Find some pseudoinverse V to

$$A := \begin{bmatrix} 1 \\ 2 \end{bmatrix},\ B := [1, 2, 3],\ C := \begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

Here are some matrices to experiment with ..

$$\begin{pmatrix} 0 & \frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} \frac{1}{5} & \frac{2}{5} \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{14} \\ \frac{1}{7} \\ \frac{3}{14} \end{pmatrix} \quad \begin{pmatrix} 1 & -1 \\ 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} \frac{5}{9} & -\frac{4}{9} \\ \frac{2}{9} & \frac{2}{9} \\ -\frac{4}{9} & \frac{5}{9} \end{pmatrix}$$

b. Compare the type (i.e. the pattern $n \times m$) of a matrix A with that of one of its pseudoinverse V and the type of its transpose A^t?
c. Why can't one speak of '*the*' pseudoinverse V to A?

3.2.2 Exercise: Linear systems and Pseudoinverses.

Consider the 2×2 systems of linear equations:

$$\begin{bmatrix} 2 \cdot x + 3 \cdot y = 1 \\ -x + 2 \cdot y = 2 \end{bmatrix},\ \begin{bmatrix} 2 \cdot x + 3 \cdot y = 1 \\ -x - 1.5 \cdot y = 1 \end{bmatrix},\ \begin{bmatrix} 2 \cdot x + 3 \cdot y = 1 \\ -x - 1.5 \cdot y = 2 \end{bmatrix}$$

Write the LGS in matrix form $A*X = B$ and determine at least two different pseudoinverses for the associated system matrices A by *hand* & *head* and by EIGENMATH & *button*.

[21] For a proof see e.g. [14, p. 130 ff] for an explicit construction of a so-called *quasi-inverse* or [8, p. 95, proposition A].

3.2.3 Exercise: *mp*i as a Pseudoinverses.

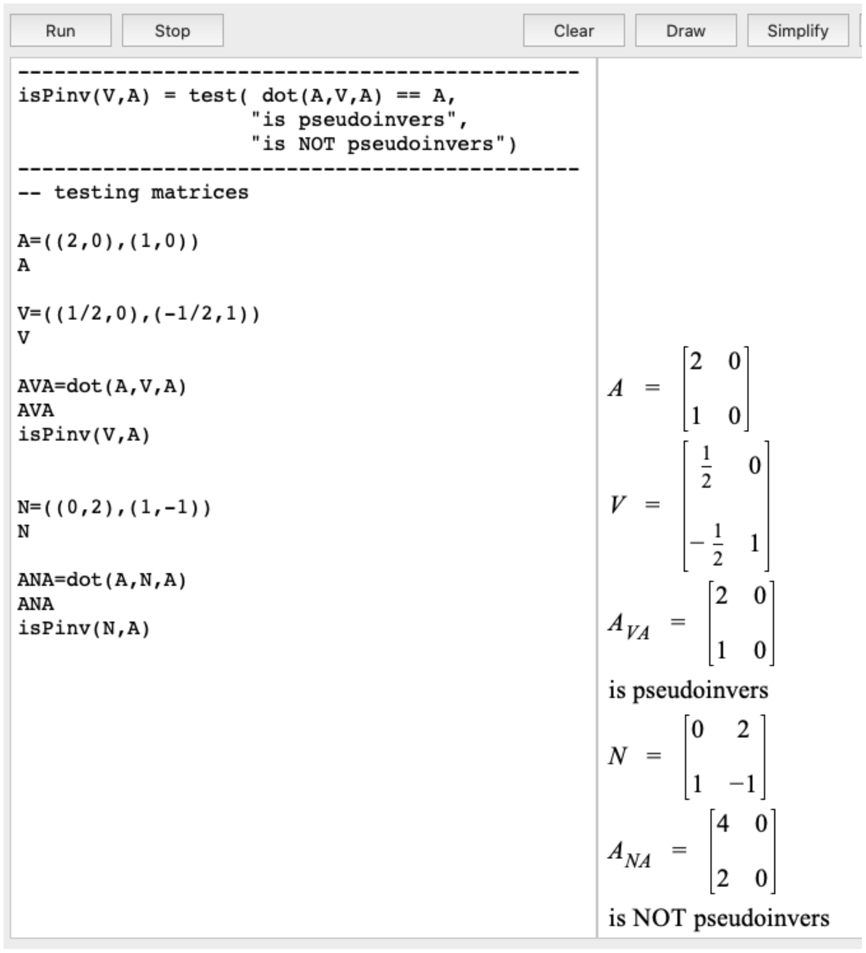
a. Show that the MOORE-PENROSE-inverse $mpi(A)$ of a matrix A - if it exists ! - is a pseudoinverse of A in the sense of definition 3.2, i.e. satisfies the equation $A * V * A = A$.
b. Calculate some examples.

3.2.4 Exercise: test of being a Pseudoinverses using EIGENMATH.

a. Define the following function in EIGENMATH to test whether a presented matrix V is a pseudoinverse to A:

```
isPinv(V,A) = test(dot(A,V,A)==A, "is pseudoinvers", "is NOT pseudoinvers")
```

and put the function in our toolbox *mpiBox*.
b. Function `isPinv(V,A)=dot(A,V,A)==A` will do the same as a. What do you say?
c. Check both EIGENMATH functions on the matrices of 3.2.1.
d. The function `isPinv` is a so called *boolean* function, which answers to the question '**is** V a pseudoinverse to A?' with '*true*=1' or '*false*=0'.
Explain looking at `isPinv(V,A)` and `isPinv(N,A)`.
Hint.



```
-----------------------------------------------
isPinv(V,A) = test( dot(A,V,A) == A,
                    "is pseudoinvers",
                    "is NOT pseudoinvers")
-----------------------------------------------
-- testing matrices

A=((2,0),(1,0))
A

V=((1/2,0),(-1/2,1))
V

AVA=dot(A,V,A)
AVA
isPinv(V,A)


N=((0,2),(1,-1))
N

ANA=dot(A,N,A)
ANA
isPinv(N,A)
```

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

$$V = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix}$$

$$A_{VA} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix}$$

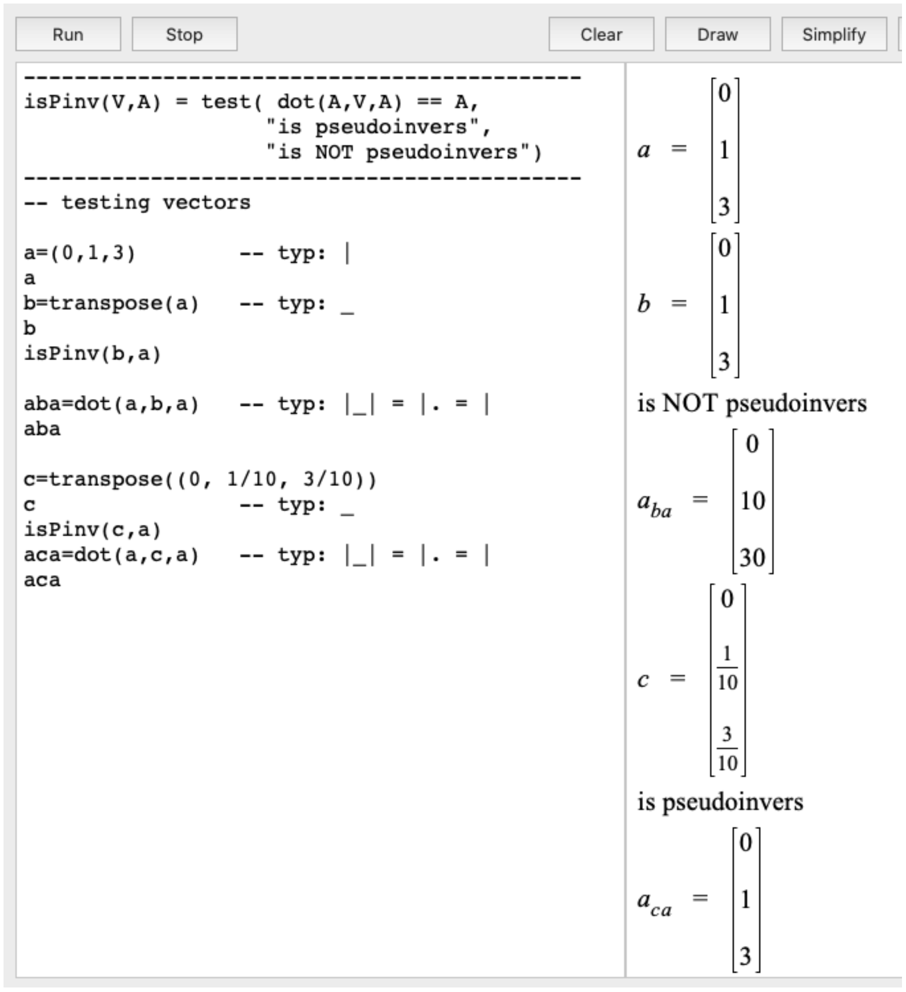
is pseudoinvers

$$N = \begin{bmatrix} 0 & 2 \\ 1 & -1 \end{bmatrix}$$

$$A_{NA} = \begin{bmatrix} 4 & 0 \\ 2 & 0 \end{bmatrix}$$

is NOT pseudoinvers

.. vectors:



```
-------------------------------------------
isPinv(V,A) = test( dot(A,V,A) == A,
                    "is pseudoinvers",
                    "is NOT pseudoinvers")
-------------------------------------------
-- testing vectors

a=(0,1,3)         -- typ: |
a
b=transpose(a)    -- typ: _
b
isPinv(b,a)

aba=dot(a,b,a)    -- typ: |_| = |. = |
aba

c=transpose((0, 1/10, 3/10))
c                 -- typ: _
isPinv(c,a)
aca=dot(a,c,a)    -- typ: |_| = |. = |
aca
```

$$a = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

is NOT pseudoinvers

$$a_{ba} = \begin{bmatrix} 0 \\ 10 \\ 30 \end{bmatrix}$$

$$c = \begin{bmatrix} 0 \\ \frac{1}{10} \\ \frac{3}{10} \end{bmatrix}$$

is pseudoinvers

$$a_{ca} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

3.3 Solvability of Linear Systems.

We are now exploring the usefulness of the concept of a pseudoinverse and are first looking for a criterion for the solvability of any (in particular singular or *over*determined) LS. Then we look for a explicit formula for the complete solution set of the LS.

If any $m \times n$-LS $A * X = B$ is given, the following alternative applies:

- If A is **invertible**, the solution set of the LS $A * X = B$ is given explicitly in the form $X = A^{-1} * B$. The following test applies: $A * (A^{-1} * B) = B$

- If A is **not invertible**, then A is always pseudo-invertible according to 3.3.3a with V as any pseudo-inverse (e.g. often the *mp*i) and $X = V * B$ 'should desirably (analogous to first alternative) be a *special* solution.

  The following test should apply: $A * (V * B) = B$.

In fact, we the have the following theorem.

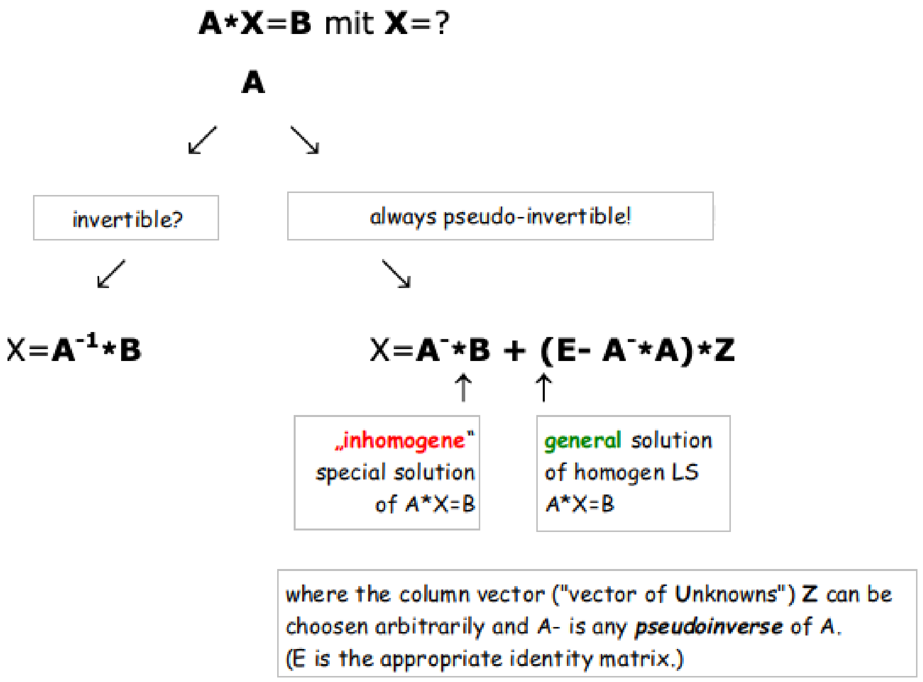
Theorem 3.1. *(solvability and totality of solutions of Linear Systems $A * X = B$)*
A. *$A * X = B$ solvable $\Leftrightarrow$ $A * A^- * B = B$ with A^- any pseudoinverse.* (LSS)[22]
B. *If A. is the case, then*[23]

$$X = A^- * B + U - A^- * A * U \qquad (LSSS)$$

is the general solution of $A * X = B$ parameterized with any matrix U.
There are no more or other solutions. Herein A^- is any choosen pseudo-inverse of A.

Proof: for yourself. □

Here is a *summary* as flow diagram.


A*X=B mit X=?
A
invertible?
always pseudo-invertible!
X=A⁻¹*B
X=A⁻*B + (E- A⁻*A)*Z
„inhomogene" special solution of A*X=B
general solution of homogen LS A*X=B
where the column vector ("vector of Unknowns") Z can be choosen arbitrarily and A- is any pseudoinverse of A.
(E is the appropriate identity matrix.)

Our goal is now to formulate the solvability criterion (LSS) and the explicit formula for the total solution set (LSSS) in the EIGENMATH syntax and thus to make it automatically calculable.

[22] LSS = Linear System Solvability
[23] LSSS = Linear System Solution Set

3.4 Eigenmath lab: Solvability criterion for Linear Systems.

We now define an Eigenmath function `isSolvable(...)`, which formulates the solvability condition (LSS) for linaer systems $A * X = B$ in Eigenmath syntax and outputs the message '*LS solvable*' in the case of solvability and '*NOT solvable*' in case of non-solvability. `isSolvable` needs the data A and B of the LS $A * X = B$ as well as any pseudoinverse P for A as inputs:

```
isSolvable(A,B,P) = test( -- LSS criterion
                          and( dot(A,P,A)==A, dot(A,P,B)==B ),
                          "is solvable",
                          "is NOT solvable")
```

a. Put the function `isSolvable` in your toolbox *mpiBox*.
b. Function

```
isSolvable1(A,B,V) =  and( dot(A,V,B)==B,  dot(A,V,A)==A ) --(LSS)
```

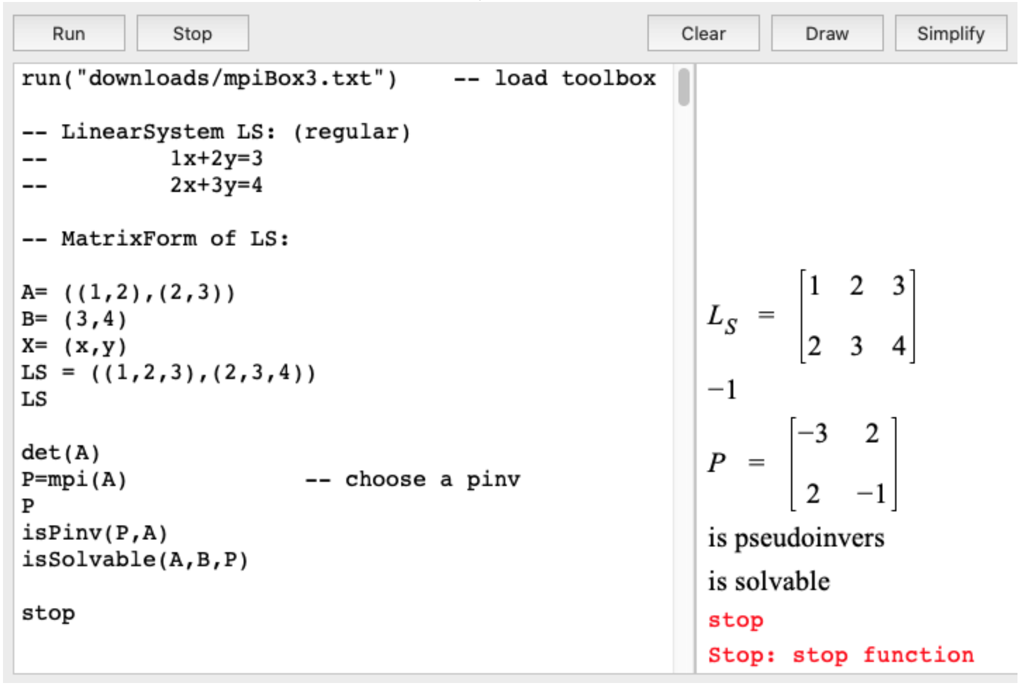
will do the same as a. Which version do you prefer? Why?

3.4.1 Exercise: Solvability criterion for Linear Systems.

Test the solvability of the LS from 3.1.1, 3.1.2 and 3.2.2 with the help of the solvability condition (LSS) from above without (! yes, at first ♡) and with the help of Eigenmath .

3.4.2 Example: the solvability criterion - regular LS.

Look at the regular linear system of equations `{1x + 2y = 3, 2x + 3y = 4}` .
We take our toolbox and check the solvability:



```
run("downloads/mpiBox3.txt")    -- load toolbox

-- LinearSystem LS: (regular)
--          1x+2y=3
--          2x+3y=4

-- MatrixForm of LS:

A= ((1,2),(2,3))
B= (3,4)
X= (x,y)
LS = ((1,2,3),(2,3,4))
LS

det(A)
P=mpi(A)              -- choose a pinv
P
isPinv(P,A)
isSolvable(A,B,P)

stop
```

$$L_S = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

-1

$$P = \begin{bmatrix} -3 & 2 \\ 2 & -1 \end{bmatrix}$$

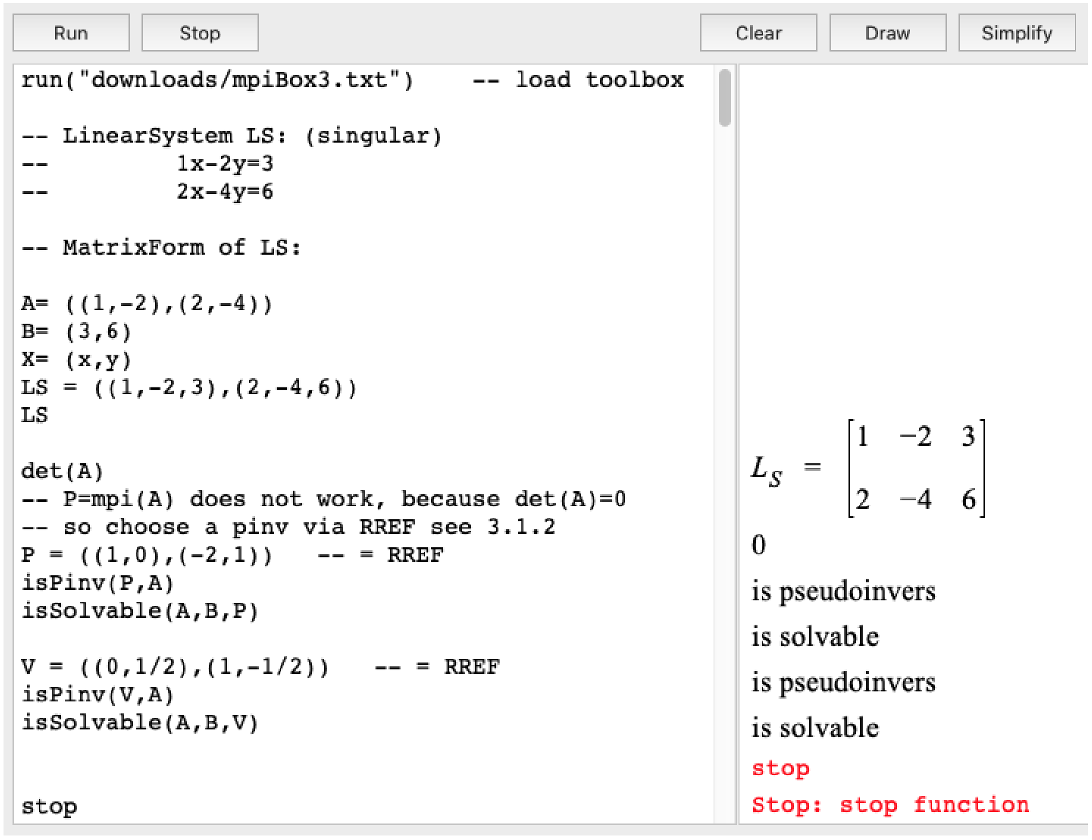
is pseudoinvers

is solvable

```
stop
Stop: stop function
```

3.4.3 Example: the solvability criterion - singular LS.

Look now at the singular linear system of equations `{1x - 2y = 3, 2x - 4y = 6}` . We take our toolbox and check the solvability:



```
run("downloads/mpiBox3.txt")    -- load toolbox

-- LinearSystem LS: (singular)
--          1x-2y=3
--          2x-4y=6

-- MatrixForm of LS:

A= ((1,-2),(2,-4))
B= (3,6)
X= (x,y)
LS = ((1,-2,3),(2,-4,6))
LS

det(A)
-- P=mpi(A) does not work, because det(A)=0
-- so choose a pinv via RREF see 3.1.2
P = ((1,0),(-2,1))   -- = RREF
isPinv(P,A)
isSolvable(A,B,P)

V = ((0,1/2),(1,-1/2))   -- = RREF
isPinv(V,A)
isSolvable(A,B,V)


stop
```

$$L_S = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -4 & 6 \end{bmatrix}$$

```
0
is pseudoinvers
is solvable
is pseudoinvers
is solvable
stop
Stop: stop function
```

Because of the singularity of the system, we have $det(A) = 0$ and the MOORE-PENROSE pseudoinverse does not exists. Therefore one has to construct a pseudoinverse for oneself using e.g. the RREF process. We had done this before for this LS in 3.1.2. We choose happily this result: $P = RREF = \left(\begin{smallmatrix} 1 & 0 \\ -2 & 1 \end{smallmatrix}\right)$. With this pseudoinverse the test succeeds.
We could also choose another pinv e.g. $V = \left(\begin{smallmatrix} 0 & 0.5 \\ 1 & -0.5 \end{smallmatrix}\right)$ with the same successful check for solvability.

Now we want to see the solution set of such an LS! Here we are:

3.5 Eigenmath code: complete solution set of Linear Systems.

Let's now define an Eigenmath function `solSet(...)`[24], which executes formula (LSSS) of theorem 3.1 in section 3.3 to determine the complete solution set of the linear system $A * X = B$ in Eigenmath syntax.

`solSet` needs as inputs
- the data matrices A and B of the LS $A * X = B$
- *any* pseudoinverse P for A
- a choice for the LS system variables X
and outputs the solution set of the LS using the variable setting from X.

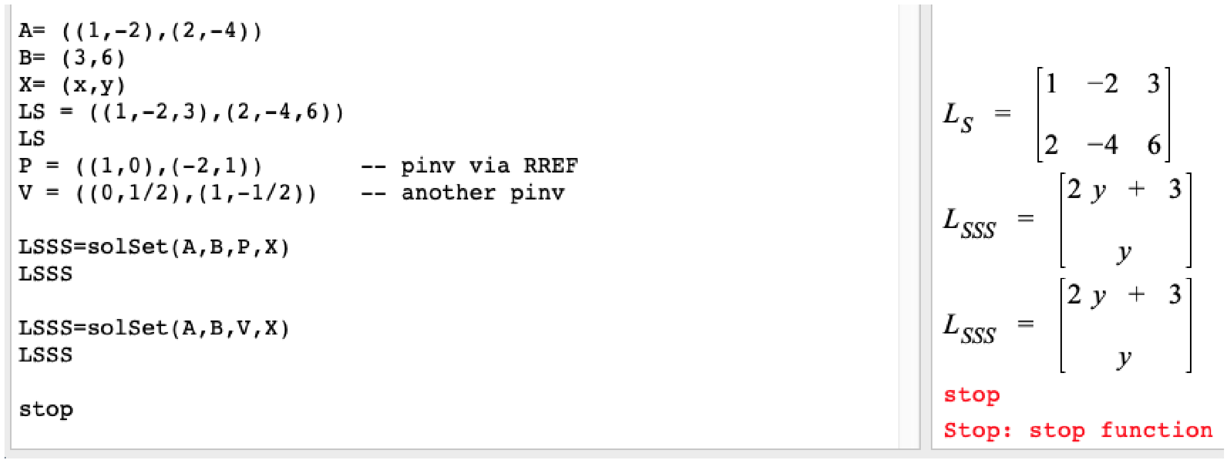
```
solSet(A,B,P,X) =  do(
                   m=dim(dot(P,A),1),          -- (1)
                   n=dim(dot(P,A),2),          -- (2)
                   E1=unit(m,n),               -- (3)
                   dot(P,B) + dot( (E1-dot(P,A)), X)      --(4) is (LSSS)
                   )
```

Remark. First we read off the row resp. column dimensions of the product $P * A$ in (1) and (2) to form the suitable unity (or identity) matrix $E1$ in (3). Using $E1$ we can do the subtraction of now equal typ matrices $E1 - P * A$ and then build the product $(E1 - P * A) * X$ in (4).

Please put the code of `solSet` in the toolbox `mpiBox` before running the next exercise.

3.5.1 Eigenmath Exercise: Solution set of a singular Linear System.

Let's construct the complete solution set of the singular linear system of equations `{1x - 2y = 3, 2x - 4y = 6}` from section 3.4.3. We have:



```
A= ((1,-2),(2,-4))
B= (3,6)
X= (x,y)
LS = ((1,-2,3),(2,-4,6))
LS
P = ((1,0),(-2,1))       -- pinv via RREF
V = ((0,1/2),(1,-1/2))   -- another pinv

LSSS=solSet(A,B,P,X)
LSSS

LSSS=solSet(A,B,V,X)
LSSS

stop
```

$$L_S = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -4 & 6 \end{bmatrix}$$

$$L_{SSS} = \begin{bmatrix} 2y + 3 \\ y \end{bmatrix}$$

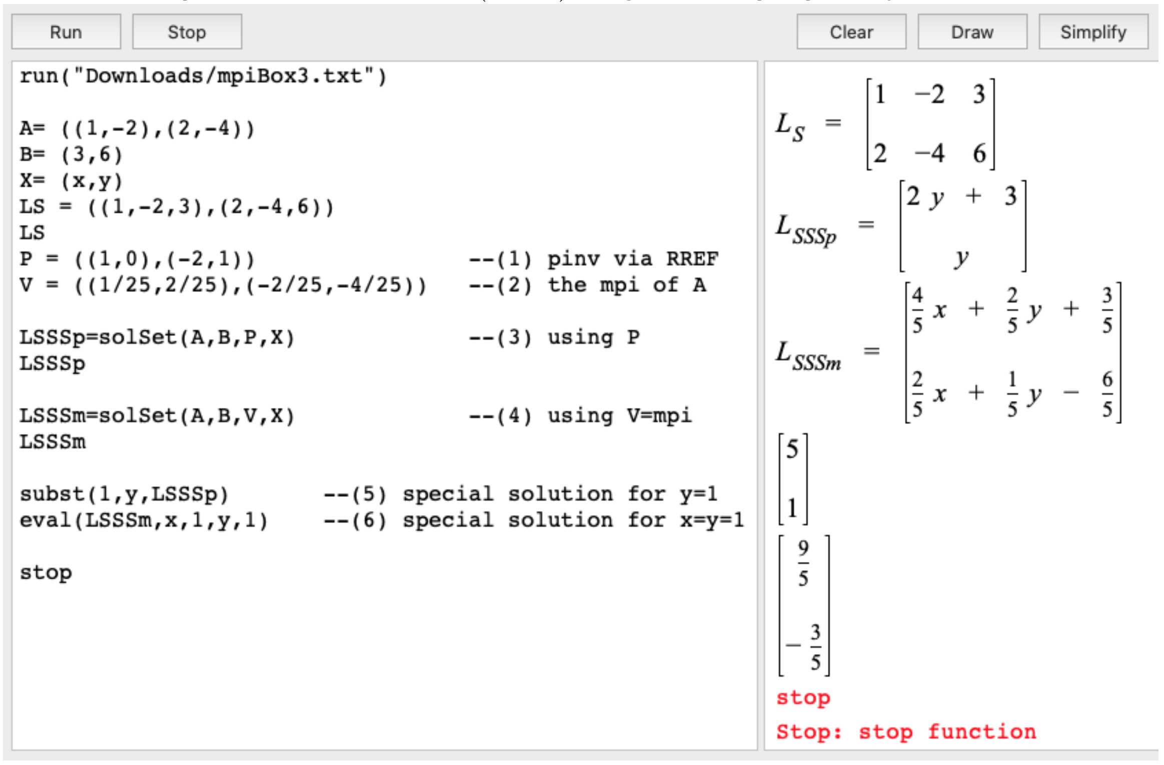
$$L_{SSS} = \begin{bmatrix} 2y + 3 \\ y \end{bmatrix}$$

```
stop
Stop: stop function
```

[24]`solSet` = *sol*ution set

We see that the full solution set LSSS is a straight line in $\mathbb{R}^2$ with the parametric vector equation $g : (2y + 3, y)$. For $y = 1$ the special solution point is $Q = (5, 1)$ on g.

If we take as pseudoinverse V the *mpi* (see (2)), then we get another parametrization in LSSSm. Here we get a special solution (point) on g choosing e.g.$x = y = 1$:



```
run("Downloads/mpiBox3.txt")

A= ((1,-2),(2,-4))
B= (3,6)
X= (x,y)
LS = ((1,-2,3),(2,-4,6))
LS
P = ((1,0),(-2,1))                --(1) pinv via RREF
V = ((1/25,2/25),(-2/25,-4/25))   --(2) the mpi of A

LSSSp=solSet(A,B,P,X)             --(3) using P
LSSSp

LSSSm=solSet(A,B,V,X)             --(4) using V=mpi
LSSSm

subst(1,y,LSSSp)        --(5) special solution for y=1
eval(LSSSm,x,1,y,1)     --(6) special solution for x=y=1

stop
```

$$L_S = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -4 & 6 \end{bmatrix}$$

$$L_{SSSp} = \begin{bmatrix} 2y + 3 \\ y \end{bmatrix}$$

$$L_{SSSm} = \begin{bmatrix} \frac{4}{5}x + \frac{2}{5}y + \frac{3}{5} \\ \frac{2}{5}x + \frac{1}{5}y - \frac{6}{5} \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{9}{5} \\ -\frac{3}{5} \end{bmatrix}$$

```
stop
Stop: stop function
```

The following figure 4. shows the full solution set LSSS= g:

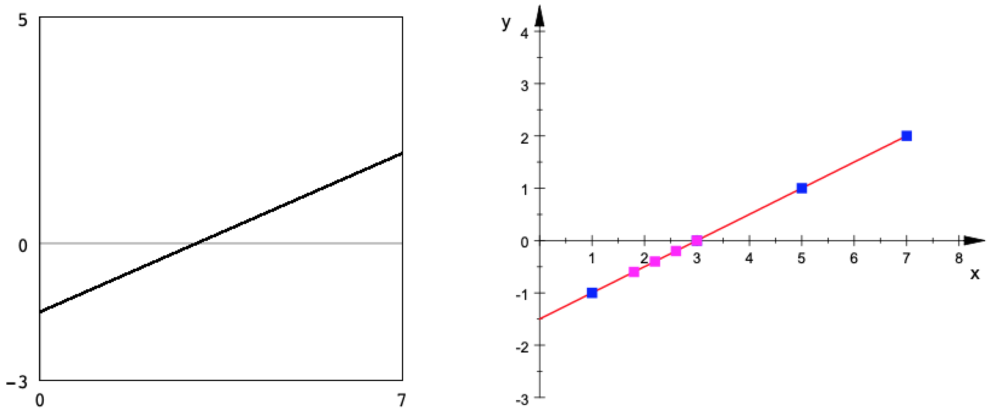


Figure 4: Representation of solutions LSSS of $\{1x - 2y = 3, 2x - 4y = 6\}$ as graph (red) of function $f : x \mapsto \frac{1}{2}x - \frac{3}{2}$ and as point set (blue) of all $(2y + 3, y)$ and of all points (magenta) with $(4/5 * x + 2/5 * y + 3/5, 2/5 * x + 1/5 * y - 6/5)$.

1. Show, that the solution set LSSS is equivalent to the graph of the function $x \mapsto \frac{1}{2}x - \frac{3}{2}$.
2. Let EIGENMATH calculate the coordinates of the blue points and the magenta point $(3, 0)$. Hint: section 1.1.11 and `for(y,-1,2, print(..)  )`
3. Figure 4.a on the left was drawn by EIGENMATH with the commands

   ```
   xrange = (0,7)
   yrange = (-3,5)
   draw(x/2-3/2)
   ```

   Let EIGENMATH draw the representation (2t+3,t) of LSSS $= g$.
4. Proof that the solution set representations LSSSp and LSSSm describe the *same* set.

3.6 Lexicon: Math vs. EIGENMATH

Lexicon: Math vs. EIGENMATH

	mathematical notation	EIGENMATH notation
10	V is pseudoinvers to A $\therefore$ A*V*A=A	`isPinv : dot(A,V,A)==A`
11	$A * X = B$ solvable with V as pseudoinvers $\therefore A * V * B = B$	`isSolvable : dot(A,V,B)==B`
12	full solution set of $A * X = B$ with pinv P & variables Z: $P * B + (E - P * A) * Z$	`solSet(A,B,P,Z)`

3.7 Problem.

Given is the following 3×3 LS

$$\begin{bmatrix} 1 \cdot x + 2 \cdot y + 3 \cdot z = 4 \\ 3 \cdot x + 4 \cdot y + 3 \cdot z = 2 \\ 6 \cdot x + 8 \cdot y + 6 \cdot z = 4 \end{bmatrix}$$

1. What is A, B and the LS of the system?
2. Build a pseudoinverse P of A and check if `isPinv(P,A)` is true.
   Hint: use Elementary matrices $Em()$ with $n = 3$. The final state of RREF should be

$$\begin{pmatrix} 1 & 0 & -3 & -6 \\ 0 & 1 & 3 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

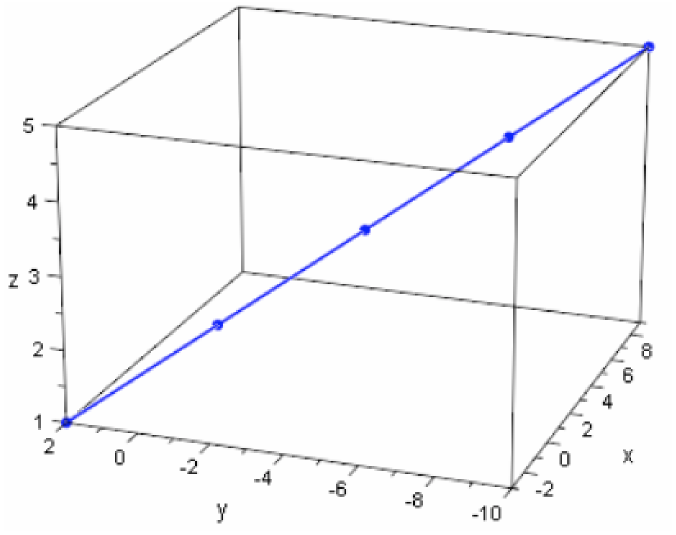
3. Verify: The pseudoinverse P of a) as product of the Em is $P = RREF =$
$$\begin{pmatrix} -2 & 1 & 0 \\ \frac{3}{2} & -\frac{1}{2} & 0 \\ 0 & -2 & 1 \end{pmatrix}$$

4. Construct a pseudoinverse V of A using the ansatz $(A|E)$ with uinty matrix E appropriate. Hint: the final state should be
$$\begin{pmatrix} 1 & 0 & -3 & -2 & 0 & \frac{1}{2} \\ 0 & 1 & 3 & \frac{3}{2} & 0 & -\frac{1}{4} \\ 0 & 0 & 0 & 0 & 1 & -\frac{1}{2} \end{pmatrix}$$

5. Let EIGENMATH check the solvability of LS and let it determine the full solution set.

   Hint: the `solSet` answer should be
$$[x = 3 \cdot z - 6,\ y = 5 - 3 \cdot z]$$

6. The following figure shows a visualisation of the solution set in $\mathbb{R}^3$.



   - Calculate the coordinates of the blue points on the line using the `for` command.
   - Which z-value gives the point $(-6, 5, 0)$ on the solution line?

7. Can you use the $mpi(A)$ as a choice for a pseudoinverse to get the solution set LSSS?

4 Construction of the Moore-Penrose-Inverse

In the last chapter we used pseudoinverses to solve systems of e.g. overdetermined linear equations. These pseudoinverses had been produced by means of the Gauss-Jordan algorithm and we saved the whole solution process in the matrix RREF i.e. a pseudoinverse. If we use these 'handmade' pseudoinverses, the representation of the solution set was often more aesthetic resp. simple:

$$\begin{pmatrix} 0 & \frac{1}{2} \\ 1 & -\frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} 2 \cdot y + 3 \\ y \end{pmatrix} \qquad\qquad \begin{pmatrix} \frac{1}{25} & \frac{2}{25} \\ -\frac{2}{25} & -\frac{4}{25} \end{pmatrix} \quad \begin{pmatrix} \frac{4 \cdot x}{5} + \frac{2 \cdot y}{5} + \frac{3}{5} \\ \frac{2 \cdot x}{5} + \frac{y}{5} - \frac{6}{5} \end{pmatrix}$$

Figure 5: Two representation of the solution set of $\{1x - 2y = 3, 2x - 4y = 6\}$. The left pair (pseudoinverse, solSet) shows a more simple parametrized solution set compared to the pair on the right, where the pseudoinverse is *the mpi* of the system matrix.

Nevertheless we often are interested in an automatic calculation of a special pseudoinverse, *the* Moore-Penrose-inverse `mpi`, e.g. for solving best fit problems. Here we have to live with it's crooked values, because this results from distinct measurements with standardized scales and are therefore in principle unavoidable.

In this chapter we present therefore two more constructions ot the Moore-Penrose-inverse *mpi*:
- an analytic-numeric approximative approach and
- the iterative, constructive and exact Greville algorithm.

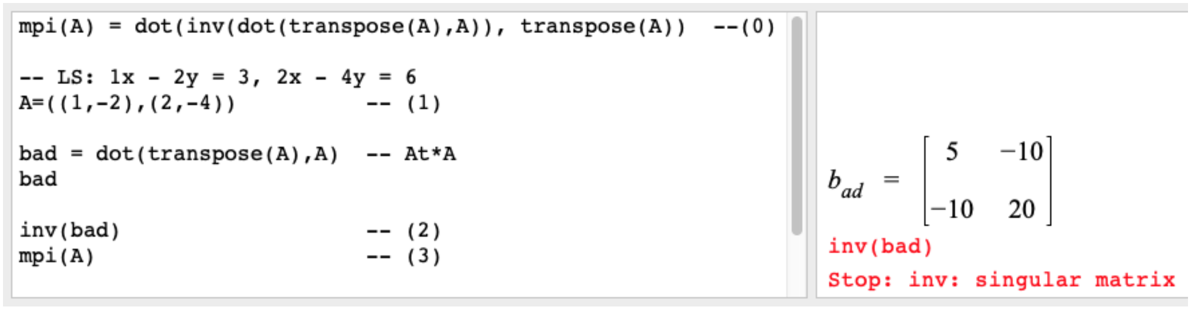
4.1 Approximative calculation of the Moore-Penrose-inverse

We ask: Is it not possible to somehow rescue the old definition $mpi(A) = (A^t * A)^{-1} * A^t$, to which we have become so used and which worked so well in many practical cases? But how to get a grip on the problem with the no-invertibility of $A^t * A$ in the defining term of *mpi* as easily as possible?
OK, here follows an idea that works - but you have to remember the calculation of limit values. Because there are no clear boundaries in mathematics we change for a moment from doing algebra and geometry to the subject of calculus. So how about this ..

4.1.1 A numerical version of the Moore-Penrose-inverse

Let's again study the Linear System $\{1x - 2y = 3, 2x - 4y = 6\}$. There was the bad guy term $A^t * A$ that caused problems and hindered to get the *mpi* as a pseudoinverse in the determination of the solution set, therefore forcing us to dodge to the RREF algorithm. We look back a bit ..

Step **1** In this first EIGENMATH session we recap the definition of the *mp*i in (0) and convince ourself that the bad term part $A^t * A$ of the *mpi* formula is not invertible (2). Therefore *mpi* can not be calculated this way



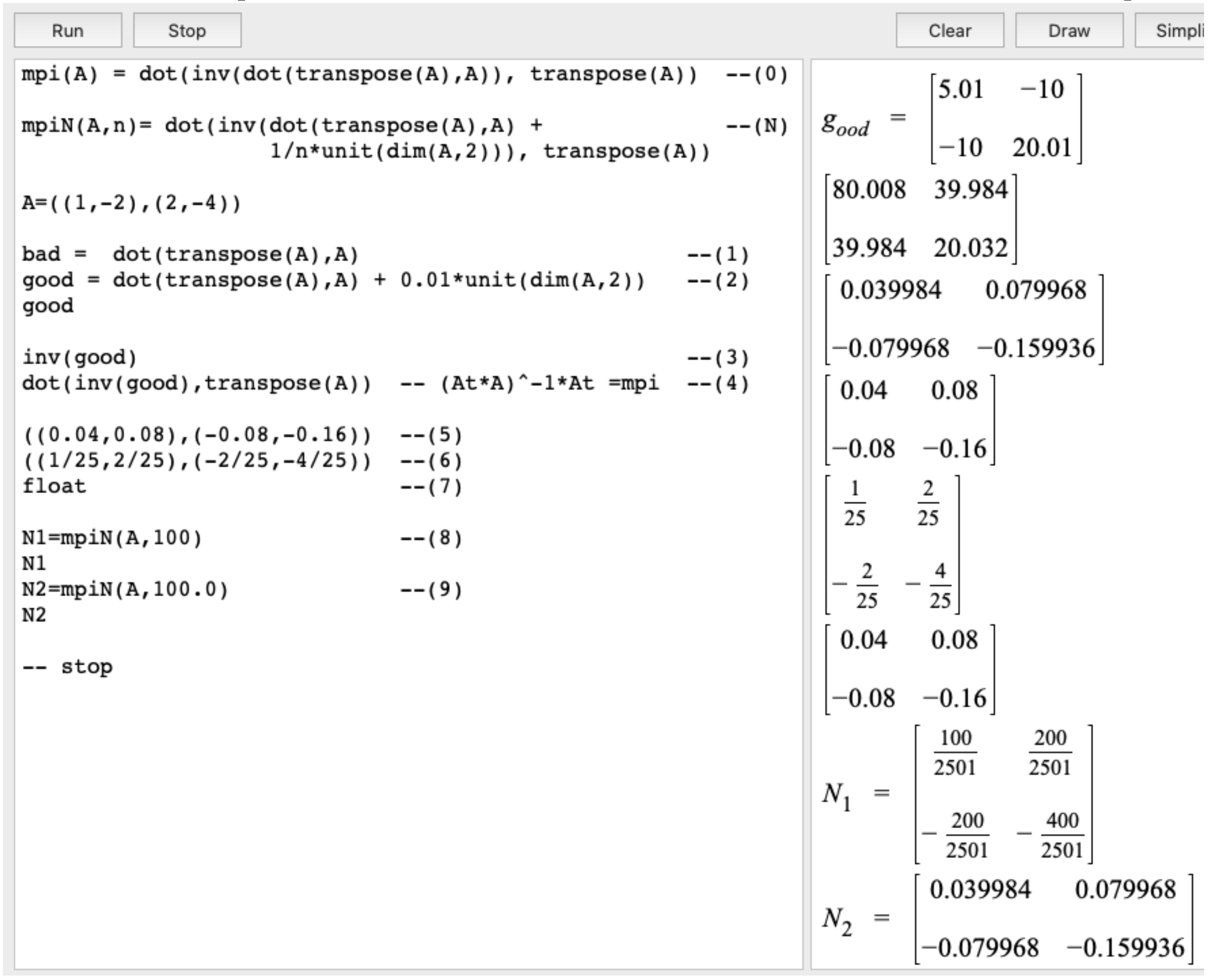
```
mpi(A) = dot(inv(dot(transpose(A),A)), transpose(A))  --(0)

-- LS: 1x - 2y = 3, 2x - 4y = 6
A=((1,-2),(2,-4))          -- (1)

bad = dot(transpose(A),A)  -- At*A
bad

inv(bad)                   -- (2)
mpi(A)                     -- (3)
```

$$b_{ad} = \begin{bmatrix} 5 & -10 \\ -10 & 20 \end{bmatrix}$$

```
inv(bad)
Stop: inv: singular matrix
```

Step **2** Let's dive into a second EIGENMATH session to make some experiments. Enjoy ♡

EIGENMATH user input: EIGENMATH output:



```
mpi(A) = dot(inv(dot(transpose(A),A)), transpose(A))  --(0)

mpiN(A,n)= dot(inv(dot(transpose(A),A) +              --(N)
                1/n*unit(dim(A,2))), transpose(A))

A=((1,-2),(2,-4))

bad =  dot(transpose(A),A)                          --(1)
good = dot(transpose(A),A) + 0.01*unit(dim(A,2))    --(2)
good

inv(good)                                           --(3)
dot(inv(good),transpose(A))  -- (At*A)^-1*At =mpi   --(4)

((0.04,0.08),(-0.08,-0.16))  --(5)
((1/25,2/25),(-2/25,-4/25))  --(6)
float                        --(7)

N1=mpiN(A,100)               --(8)
N1
N2=mpiN(A,100.0)             --(9)
N2

-- stop
```

$$g_{ood} = \begin{bmatrix} 5.01 & -10 \\ -10 & 20.01 \end{bmatrix}$$

$$\begin{bmatrix} 80.008 & 39.984 \\ 39.984 & 20.032 \end{bmatrix}$$

$$\begin{bmatrix} 0.039984 & 0.079968 \\ -0.079968 & -0.159936 \end{bmatrix}$$

$$\begin{bmatrix} 0.04 & 0.08 \\ -0.08 & -0.16 \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{25} & \frac{2}{25} \\ -\frac{2}{25} & -\frac{4}{25} \end{bmatrix}$$

$$\begin{bmatrix} 0.04 & 0.08 \\ -0.08 & -0.16 \end{bmatrix}$$

$$N_1 = \begin{bmatrix} \frac{100}{2501} & \frac{200}{2501} \\ -\frac{200}{2501} & -\frac{400}{2501} \end{bmatrix}$$

$$N_2 = \begin{bmatrix} 0.039984 & 0.079968 \\ -0.079968 & -0.159936 \end{bmatrix}$$

Remark. We plan an *approximately* solution for the otherwise existing 'real' *mpi* (6). So we should make the determinant $5 \cdot 20 - 10 \cdot 10 = 0$ of the bad term (1) at least *a little bit different from zero* without changing the original data too much. Therefore, we only change the elements on the main diagonal of $A^t * A$ by adding of a tiny scaled identity matrix of the same type as $A^t * A$: this gives the good brave term (2), which can be inverted (3) and the corresponding *mp*i is approximately calculated in (4) ♡!
Term (2) i.e. $A^t * A + o.o1 \cdot \binom{1\ 0}{0\ 1} \approx A^t * A$, is abstracted into definition (N) of a **N***early* or **N***umerical* version `mpiN` of the *mpi*. Invocations of `mpiN` in (8) and (9) with suitable choices of n suggests witch guess should give the term of the real *mpi*. One should test this guess with help of the EIGENMATH function `isPinv` from our *mpiBox.txt*.

4.1.2 the numerical version `mpiN` - exercise 1.

a. Guess a term for the *mpi* for the linear system in problem 3.7 with the help of `mpiN(A,?)`.
b. Check your choice with `isPinv`.
c. Determine the full solution set of the LS with the EIGENMATH function `solSet(.)` and the *mpi* from a. as a pseudoinverse in the call of *solSet*.
d. Proof that the representations of the solution set in 3.7 and in c. describe the same set.

4.1.3 the numerical version `mpiN` - exercise 2.

$$A := \begin{bmatrix} 1 & 2 & 1 & -2 \\ 4 & 1 & 3 & 1 \end{bmatrix}$$

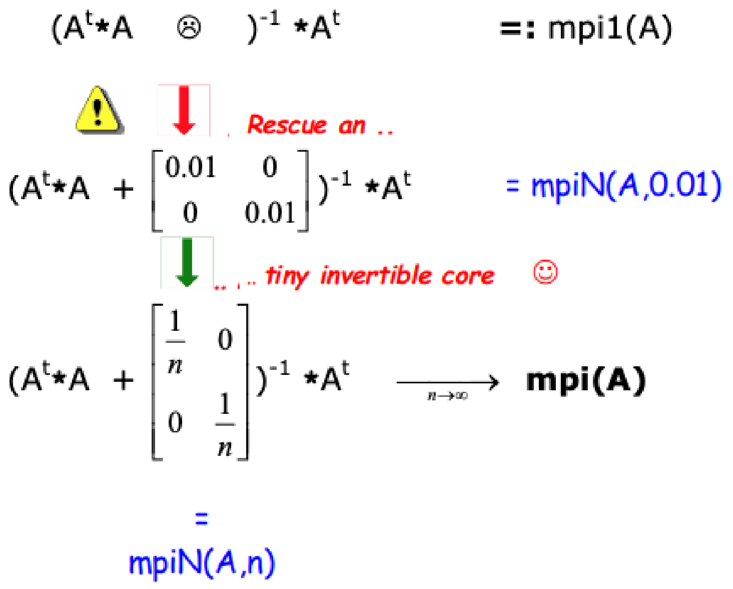
a. Calculate for the matrix A the values of the simplified version `mpiN(A,n)` of the real, but unknown *mpi* for $n = 1..5$. Make a guess for the *mpi*.
b. Study the results. Experiment. Check.

4.1.4 the numerical version `mpiN` - exercise 3.

$$A := \begin{bmatrix} 2 & 3 \\ -1 & -1.5 \end{bmatrix}$$

Calculate for the matrix A the matrix sequence `mpiN(A, 2n)` for $n = -10, -6, -4, 10$. Make a good guess for *mpi*(A).

4.1.5 the numerical version `mpiN` - flow chart summary:



$(A^t * A \quad ☹ \quad)^{-1} * A^t$ =: mpi1(A)
Rescue an ..
$(A^t * A + \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix})^{-1} * A^t$ = mpiN(A,0.01)
.. tiny invertible core ☺
$(A^t * A + \begin{bmatrix} \frac{1}{n} & 0 \\ 0 & \frac{1}{n} \end{bmatrix})^{-1} * A^t \xrightarrow[n\to\infty]{}$ mpi(A)
=
mpiN(A,n)

Remark. EIGENMATH currently has no `Limit` command. Otherwise one would have:

If A is any matrix and E is the identity matrix of the same type as $A^t * A$,
Then we define the generalized (pseudo) MOORE-PENROSE-inverse by

$$mpi(A) := \lim_{n\to\infty} (A^t * A + \tfrac{1}{n} \bullet E)^{-1} \bullet A^t$$

In EIGENMATH this definition would then be an executable *exact* formula for the *mpi*.

We now need a check to test, if a calculated matrix P is *the* MOORE-PENROSE-inverse of a given matrix A.

4.2 Definition: MOORE-PENROSE-inverse.

The matrix $\overset{n\times m}{P}$ of typ $n \times m$ is **the** MOORE-PENROSE-inverse of the matrix $\overset{m\times n}{A}$ of typ $m \times n$, if the following 4 conditions are fullfiled:

$$A * P * A = A \qquad (4.1)$$
$$P * A * P = P \qquad (4.2)$$
$$(P * A)^t = P * A \qquad (4.3)$$
$$(A * P)^t = A * P \qquad (4.4)$$

Remark.

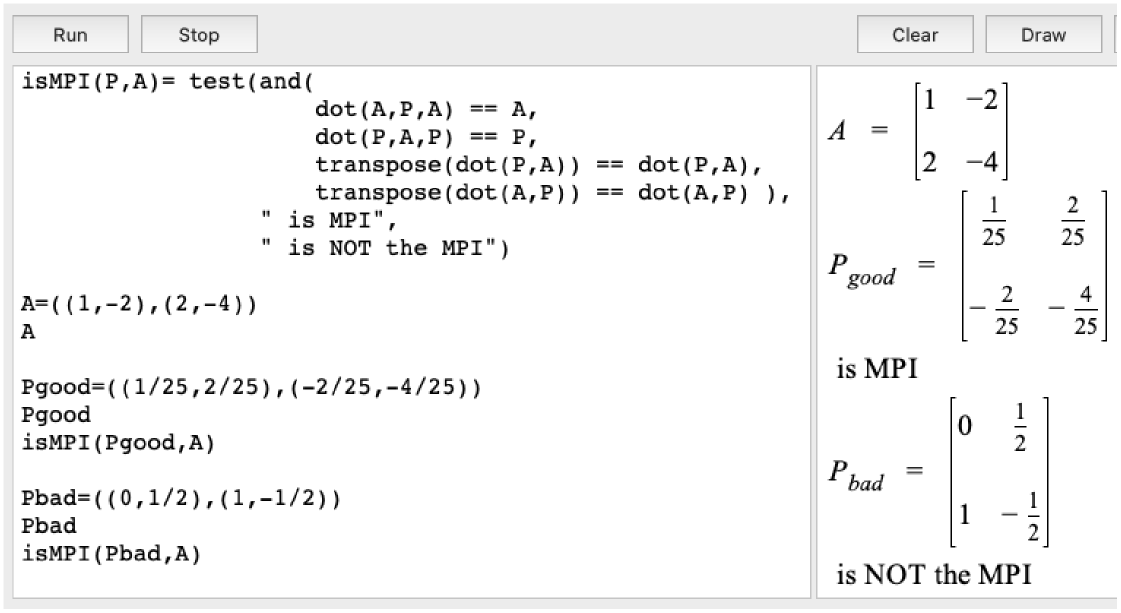
1. This unique MOORE-PENROSE-inverse of A is noted as A^+ (read: 'A plus').
2. There is an unique MOORE-PENROSE-inverse for *every* matrix (vector, too).
3. Condition (4.1) means that $\overset{n\times m}{P} * \overset{m\times n}{A} = \overset{n\times n}{E}$ ist the unitiy matrix E, i.e. P is a *left*-inverse for A. P is - because of (4.2) - also a *right*-inverse for A.

We formulate the 4 conditions as a boolean function in EIGENMATH :

```
isMPI(P,A)= test(and(
                     dot(A,P,A) == A,
                     dot(P,A,P) == P,
                     transpose(dot(P,A)) == dot(P,A),
                     transpose(dot(A,P)) == dot(A,P) ),
                " is MPI",
                " is NOT the MPI")
```

4.2.1 Example: testing a matrix of being the MOORE-PENROSE-inverse.

Please put the definition `isMPI` in your *mpiBox* or copy it in a fresh EIGENMATH session. Then we have e.g. for the matrix $A = \left(\begin{smallmatrix}1 & -2\\ 2 & -4\end{smallmatrix}\right)$ from section 4.1.1:



```
isMPI(P,A)= test(and(
                     dot(A,P,A) == A,
                     dot(P,A,P) == P,
                     transpose(dot(P,A)) == dot(P,A),
                     transpose(dot(A,P)) == dot(A,P) ),
                " is MPI",
                " is NOT the MPI")

A=((1,-2),(2,-4))
A

Pgood=((1/25,2/25),(-2/25,-4/25))
Pgood
isMPI(Pgood,A)

Pbad=((0,1/2),(1,-1/2))
Pbad
isMPI(Pbad,A)
```

$$A = \begin{bmatrix}1 & -2\\ 2 & -4\end{bmatrix}$$

$$P_{good} = \begin{bmatrix}\frac{1}{25} & \frac{2}{25}\\ -\frac{2}{25} & -\frac{4}{25}\end{bmatrix}$$

is MPI

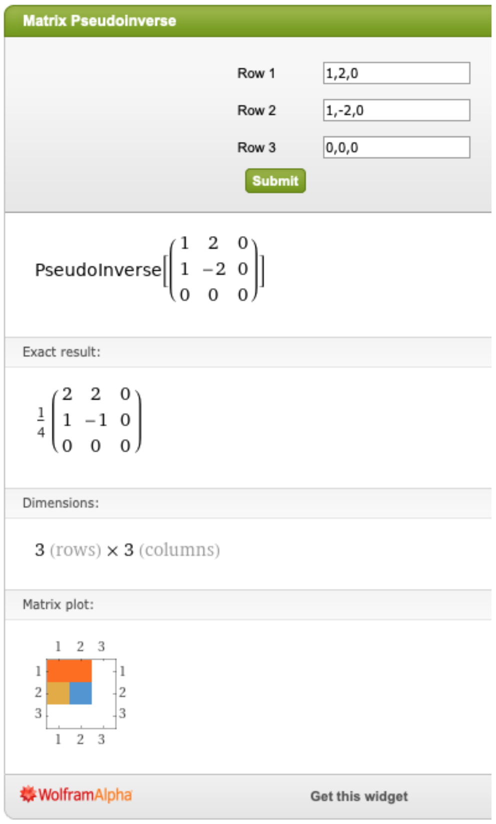
$$P_{bad} = \begin{bmatrix}0 & \frac{1}{2}\\ 1 & -\frac{1}{2}\end{bmatrix}$$

is NOT the MPI

Be warned: P_{bad} is nevertheless *a pseudo*inverse of A - but NOT the *mpi*!

4.2.2 Exercise: testing a matrix of being the Moore-Penrose-inverse.

Visit the Wolfram widget gallery for calculating the *mpi* of a 3×3 matrix:[25]



Matrix PseudoInverse

Row 1	1,2,0
Row 2	1,-2,0
Row 3	0,0,0

Submit

PseudoInverse$\left[\begin{pmatrix} 1 & 2 & 0 \\ 1 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix}\right]$

Exact result:

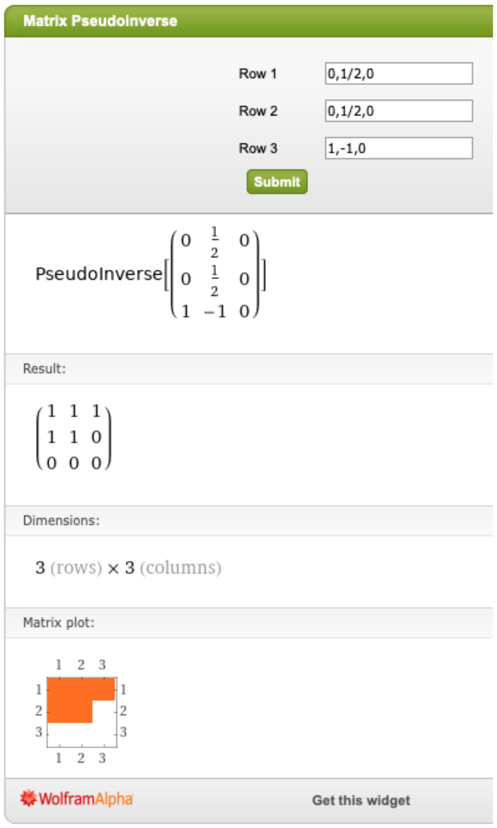
$\frac{1}{4}\begin{pmatrix} 2 & 2 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Dimensions:

3 (rows) × 3 (columns)

Matrix plot:

WolframAlpha Get this widget



Matrix PseudoInverse

Row 1	0,1/2,0
Row 2	0,1/2,0
Row 3	1,-1,0

Submit

PseudoInverse$\left[\begin{pmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 1 & -1 & 0 \end{pmatrix}\right]$

Result:

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

Dimensions:

3 (rows) × 3 (columns)

Matrix plot:

WolframAlpha Get this widget

a. Use Eigenmath function `isMPI` to check if these matrices are *mpi*'s or only pseudoinverses, i.e. `isPinv` is true.
b. Calculate the Moore-Penrose-inverse of matrix `A=((1,1,1),(1,0,1),(0,1,1))` with the widget and verify the Moore-Penrose-inverse conditions using Eigenmath function `isMPI`.

[25] `http://www.wolframalpha.com/widgets/gallery/view.jsp?id=5f520ec5d51f4746373a2bd4f857c2ed`

4.3 The Greville algorithm in Eigenmath

Now we give an iterative procedure to determine the Moore-Penrose-inverse of a given Matrix A, which terminates after finitly many steps. Therefore we view at the matrix as a list of columns. Then this column list is stepwise visited from the first *column* to the last one, while simultaneously the *mpi* matrix $\mathbf{A}^+$ is stepwise build up from the first *line* to the last line.

4.3.1 The Greville algorithm.

Greville. [26][27]

Start: let $\overset{m\times n}{A}$ be $[a_1\, a_2\, \ldots\, a_n]$ be the $m \times n$ matrix A in his column representation.

let $\overset{m\times n}{A_k}$ be $[a_1\, a_2\, \ldots\, a_k]$ be the $m \times k$ consiting of the first k columns of A.

we have $A_k = [A_{k-1}\, a_k]$

Then: do for $j \geq 2$

$d_j^t \overset{def}{=} a_j^t * (A_{j-1}^+)^t * A_{j-1}^+$

$c_j \overset{def}{=} (E - A_{j-1} * A_{j-1}^+) * a_j$

$b_j^t \overset{def}{=} c_j^+ + \frac{1-c_j^+ * c_j}{1+d_j^t * a_j} d_j^t$

we have

$$A_j^+ = [A_{j-1}\, a_j]^+ = \begin{bmatrix} A_{j-1}^+ - A_{j-1}^+ * a_j * b_j^t \\ b_j^t \end{bmatrix}$$

and for $j = 1$

$$A_1^+ = a_1^+ = \frac{1}{a_1^t * a_1} a_1^t$$

Please note:

- d_j^t is a row vector, c_j a column (therefore c_j^+ a row) and b_j^t a row.
- $A_1 = a_1$ consists of one column vector.
- we have a constructive 'count down' $d_j \triangleright c_j \triangleright b_j \rightsquigarrow a_j^+$.

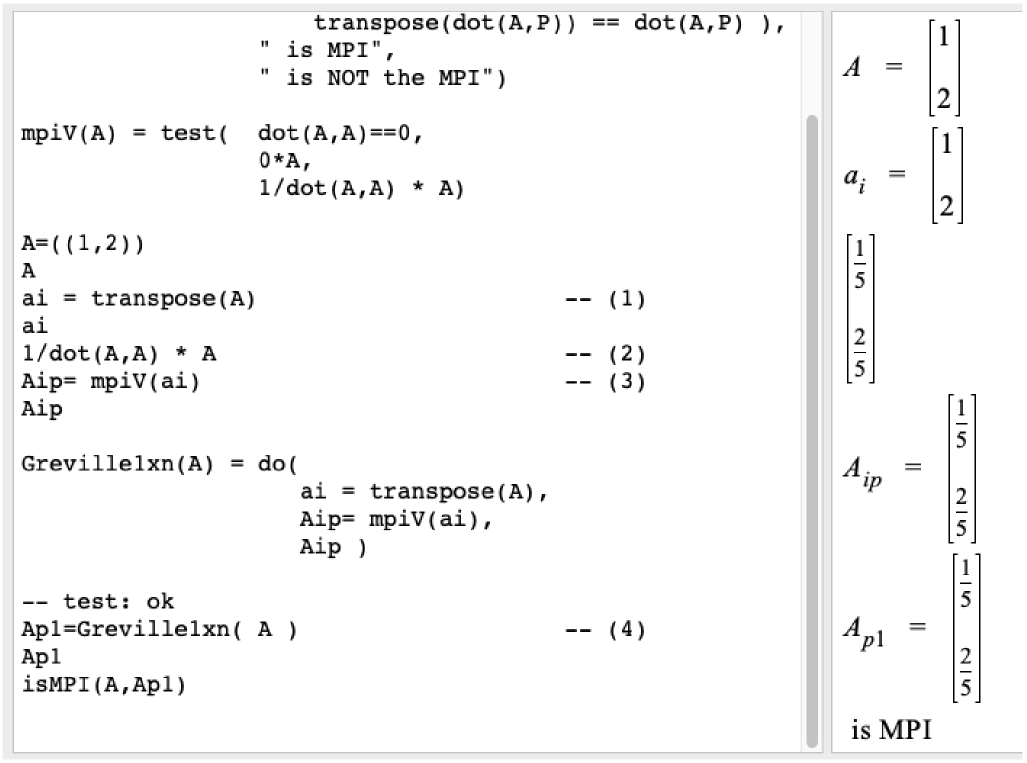
We will now study the first cases $j = 1$, $j = 2$ and $j = 3$ of the Greville algorithm to get a feeling for progamming in Eigenmath and to get used to some peculiarities.

[26] Look at the original article [6].

[27] We adopt the formulation of [17, p. 115]. Other versions of the algorithm could be found e.g. in [1], [15, p. 4] or [16, p. 3].

4.3.2 EIGENMATH case study: Greville $1 \times n$.

We study the first case $j = 1$ of the Greville algorithm. That is we are in the last line $A_1^+ = ...$ of section 4.3.1. This case is formulated in EIGENMATH in line (2) of the following screenshot. Our example matrix is $A = (1, 2)$.



```
                 transpose(dot(A,P)) == dot(A,P) ),
                 " is MPI",
                 " is NOT the MPI")

mpiV(A) = test(  dot(A,A)==0,
                 0*A,
                 1/dot(A,A) * A)

A=((1,2))
A
ai = transpose(A)                         -- (1)
ai
1/dot(A,A) * A                            -- (2)
Aip= mpiV(ai)                             -- (3)
Aip

Greville1xn(A) = do(
                   ai = transpose(A),
                   Aip= mpiV(ai),
                   Aip )

-- test: ok
Ap1=Greville1xn( A )                      -- (4)
Ap1
isMPI(A,Ap1)
```

$$A = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$a_i = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$
$$A_{ip} = \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$
$$A_{p1} = \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$

is MPI

Comment. Matrix $ai = (1 \quad 2)$ is the transpose of $A = \binom{1}{2}$ - this is *not* reflelected in the output of EIGENMATH. But you should think of it if you want to follow the calculation by paper and pencil. Function `mpiV` catches the case, where $A^t * A \overset{EigenM}{=}$ `dot(A,A)` $= 0$. Here (2) and (3) results therefore in the same value $Aip = (1/5 \quad 2/5)$. Checking by brain gives `mpi(A)*A = ` $Aip * A = (1/5 \quad 2/5) * \binom{1}{2} = \binom{1/5}{2/5} \bullet \binom{1}{2} = (1)$, which is the 1-dimensional unitiy matrix E.[28]

EIGENMATH function `Greville1xn(A)` abstracts the process of the calculation using a `do(..)` construct, which collects the suite of all commands in the correct sequence.
In contrast to the interpreted command suite (1), (2), (3) we have to separate the single commands in the `do(..)` 'compound' statement by commas '.. , ...'.

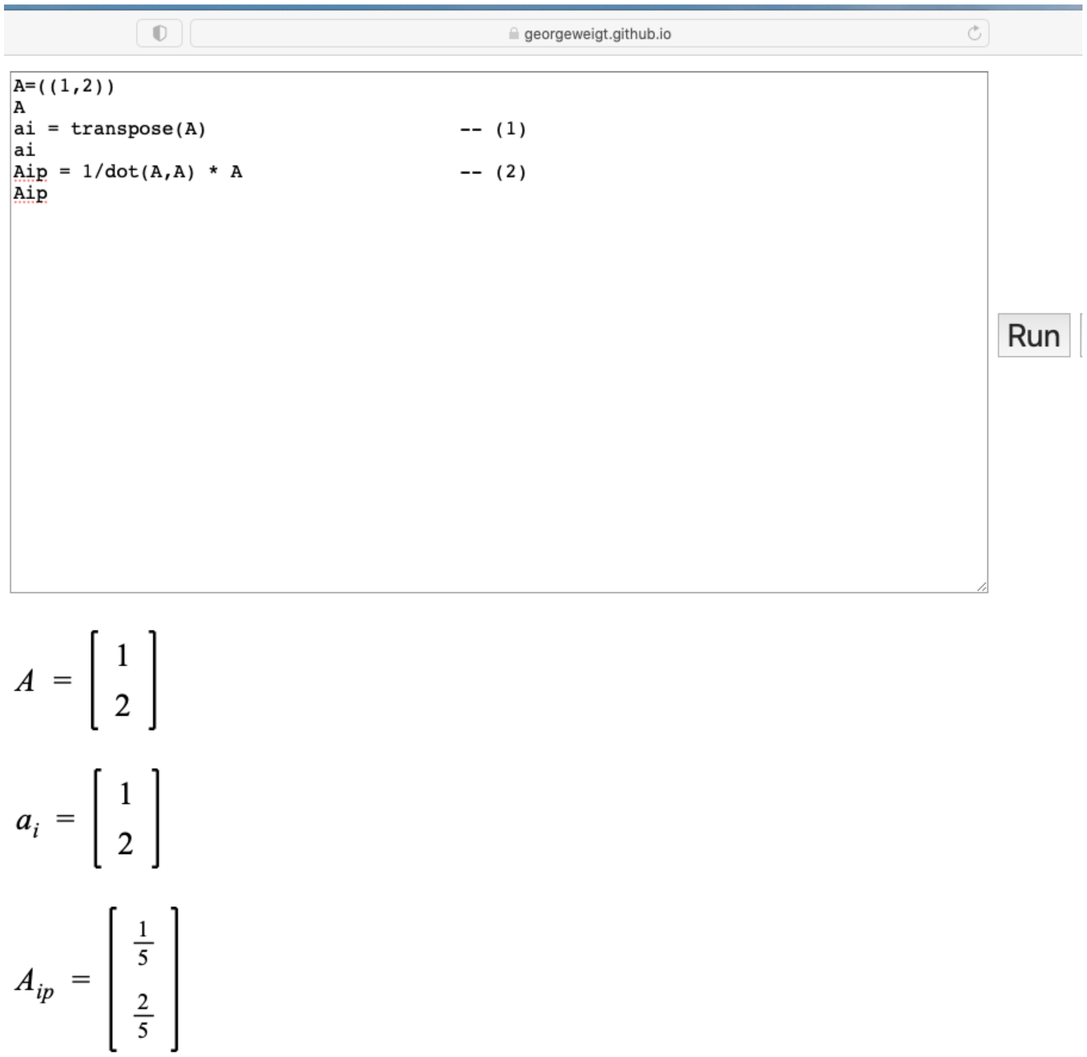
Exercise. Copy the following excerpt of the commands above into the command window of EIGENMATHs online demo[29], press the Run button and watch the output.

[28]Here $*$ denotes matrix multiplcation and $\bullet$ the scalar(dot) multiplication of vectors.
[29]https://georgeweigt.github.io/eigenmath-demo.html

EIGENMATH:

```
A=((1,2))
A
ai = transpose(A)                          -- (1)
ai
Aip = 1/dot(A,A) * A                       -- (2)
Aip
```

This should give the following situation:



georgeweigt.github.io

```
A=((1,2))
A
ai = transpose(A)                          -- (1)
ai
Aip = 1/dot(A,A) * A                       -- (2)
Aip
```

$$A = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

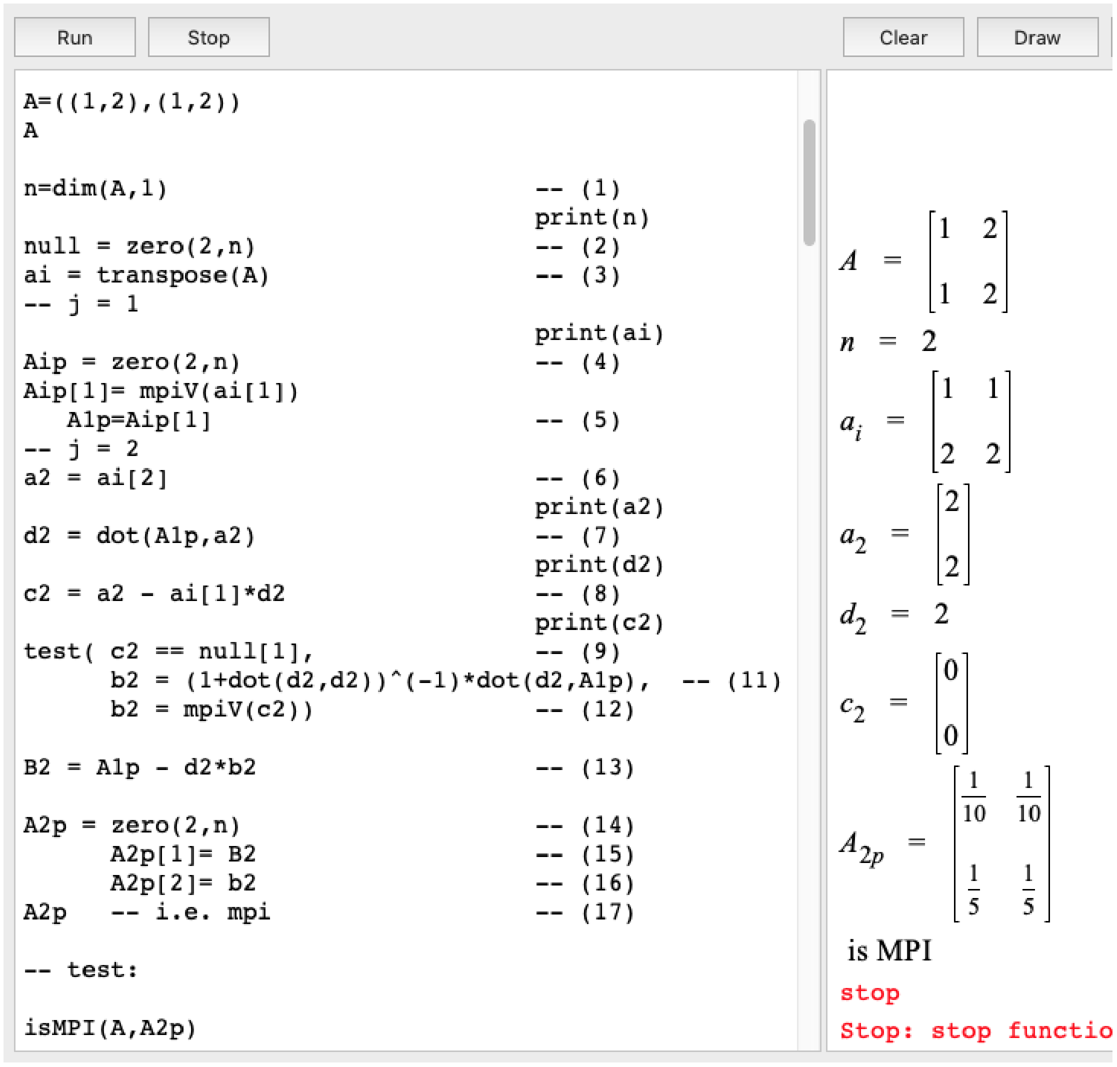
$$a_i = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$A_{ip} = \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$

a. Check using the online demo, if `Aip` is the MOORE-PENROSE-inverse of A.
b. Experiment with other 1-dimensional matrices (vectors).
Try e.g. $M = ((1, 2, 3))$ or M^t.

4.3.3 Eigenmath case study: Greville $2 \times n$.

We study the second special case $j = 2$ of the `Greville` algorithm. That is we are doing first the last line $A_1^+ = ...$ of section 4.3.1 and then follow one times the steps for calculating d_j, c_j, b_j giving A_j^+. This case is formulated in Eigenmath in lines (1) .. (17) in the following screenshot.[30] Our example matrix $A = \left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right)$ is of typ 2×2.



```
A=((1,2),(1,2))
A

n=dim(A,1)                           -- (1)
                                     print(n)
null = zero(2,n)                     -- (2)
ai = transpose(A)                    -- (3)
-- j = 1
                                     print(ai)
Aip = zero(2,n)                      -- (4)
Aip[1]= mpiV(ai[1])
   A1p=Aip[1]                        -- (5)
-- j = 2
a2 = ai[2]                           -- (6)
                                     print(a2)
d2 = dot(A1p,a2)                     -- (7)
                                     print(d2)
c2 = a2 - ai[1]*d2                   -- (8)
                                     print(c2)
test( c2 == null[1],                 -- (9)
      b2 = (1+dot(d2,d2))^(-1)*dot(d2,A1p),  -- (11)
      b2 = mpiV(c2))                 -- (12)

B2 = A1p - d2*b2                     -- (13)

A2p = zero(2,n)                      -- (14)
      A2p[1]= B2                     -- (15)
      A2p[2]= b2                     -- (16)
A2p   -- i.e. mpi                    -- (17)

-- test:

isMPI(A,A2p)
```

$$A = \begin{bmatrix}1 & 2\\1 & 2\end{bmatrix}$$

$$n = 2$$

$$a_i = \begin{bmatrix}1 & 1\\2 & 2\end{bmatrix}$$

$$a_2 = \begin{bmatrix}2\\2\end{bmatrix}$$

$$d_2 = 2$$

$$c_2 = \begin{bmatrix}0\\0\end{bmatrix}$$

$$A_{2p} = \begin{bmatrix}\frac{1}{10} & \frac{1}{10}\\ \frac{1}{5} & \frac{1}{5}\end{bmatrix}$$

```
is MPI
stop
Stop: stop functio
```

Comment. Matrix $ai = \left(\begin{smallmatrix}1&1\\2&2\end{smallmatrix}\right)$ is the transpose of the given $A = \left(\begin{smallmatrix}1&2\\1&2\end{smallmatrix}\right)$ - this is reflelected in the output (3). Line (4) defines an 'empty' container matrix, which will collect all the entries of the Moore-Penrose-inverse, named '*Aip*'.[31] This matrix `Aip` is of typ $2 \times n$, because Eigenmath `zero(.)` function does not allow to call `zero(1,..)` or

[30]♡ There is no need to copy the following lines of code, because we give a mark & copy function `Greville2xn` further down.

[31] *Aip* means A_i^+ at the end of the iteration. The intermediate matrices are called `A1p`$=A_1^+$, `A2p`$=A_2^+$ etc., see (5) and (14).

`zero(..,1)`. Therefore we use only the first entry `A1p=Aip[1]` to save the result of `mpiV` - which is a vector: `A1p =` $(1/2 \quad 1/2)$. Step $j = 1$ is ready.

Now we do step $j = 2$, i.e. we have to calculate `d2, c2, b2`.[32] In (6) we pick in `a2` the second column of matrix A. `A1p` and `a2` are both vectors, so we dot them in (7) to give the number (!) `d2`.[33] `c2` measures (8) the difference of the second column to the $d2$-fold of the first column. In (9) we check, if`c2` is `a2` - if it is true, (9) we calculate `b2` in (11) along the formula of 4.3.1 , otherwise we must use in (12) function `mpiV`. `B2` gives the first line of matrix `Aip` in (13) i.e. the first line in $A^+_j = [: \dots$ of algorithm 4.3.1. With `B2` in (15) and `b2` in (16) we have both components (17) of formula

$$A^+_{j=2} = \begin{bmatrix} A2p[1] = B2 \\ A2p[2] = b2 \end{bmatrix}$$

of algorithm 4.3.1

Exercise. a. Repeat the Eigenmath calculations (2) ... (17) with paper and pencil.
b. Do the same with matrix $M = ((0, 1/2), (0, 1/2), (1, -1))$.

4.3.4 Routine Greville2xn.

Eigenmath function `Greville2xn(A)` abstracts the process of 4.3.3 in a suite of all the commands above in *do(..)*-sequence. Beware of the comma '`,`' *after each* statement inside the round *do*-brackets (...).

```
#########################################  A  typ 2 x n
-- GREVILLE 2 x n : A1+, A2+ --> An+ = A+
-----------------------------------------  A+ typ n x 2

Greville2xn(A) = do(
   n=dim(A,1),
   m=dim(A,2),
                            --print(n),
   null = zero(2,n),
   ai = transpose(A),
                            --print(ai),
   Aip = zero(2,n),
   Aip[1]= mpiV(ai[1]),
   A1p=Aip[1],
   a2 = ai[2] ,
                            --print(a2),
   d2 = dot(A1p,a2) ,
```

[32]Because of technical considerations with respect to Eigenmath and regarding the general case of an $m \times n$ matrix, we use here a slightly modified version of the mathematical formulas in 4.3.1.

[33]This is a specialty of case $j = 2$ as we will see later and forces us to consider this case apart from the general loops $j = 3$...

```
                              --print(d2),
    c2 = a2 - ai[1]*d2,
                              --print(c2),
    test( c2 == null[1],
          b2 = (1+dot(d2,d2))^(-1)*dot(d2,A1p),
          b2 = mpiV(c2)) ,
    B2 = A1p - d2*b2 ,
    A2p = zero(2,n) ,
          A2p[1]= B2 ,
          A2p[2]= b2 ,
    A2p ) -- i.e. mpi

 ######################## END 2 x n ##################
```

Exercise. Copy the code sequence of function `Greville2xn` into the command window of EIGENMATHs online demo[34], or into the interpreter window of EIGENMATH for iMac. Do not forget to copy also the functions `isMPI(P,A)` and `mpiV(A)` if you do not use *mpiBox*.

1. Repeat the case study 4.3.3 using `Greville2xn` i.e.

   - define `A=((1,2),(1,2))` and call `Greville2xn(A)`.

   - watch some intermediate results by uncommenting the *print* statements, i.e. `--print(d2)` by deleting the 2 hyphens '`--`'.

2. Experiment with the matrices $A = ((1,2),(1,2),(3,3))$ and $A_{bad} = ((1,2,3),(1,2,3))$.

3. Look at these commands:

   ```
   M=((1,1,1),(1,1,0))
   M=transpose(M)
   M
   Mp4=Greville2xn(M)
   Mp4
   isMPI(M,Mp4)
   ```

   Which idea leads to success? Explain. Use this 'fix' to deal with matrix A_{bad} in b).[35]

4. Play with other matrices.
   Use Wolfram's pseudoinverse widget in 4.2.2 to check the results.
   Check *one* calculation by paper and pencil ♡.

[34]https://georgeweigt.github.io/eigenmath-demo.html
[35]You can follow the solution of this example in full detail in [17, p. 120 - 122].

4.3.5 Eigenmath **Exercise: Greville** $3 \times n$.

a) Write an Eigenmath function `Greville3xn(A)` using the code of `Greville2xn(..)` and the following code snippet:

```
################### Greville 3 x n ###################

Greville3xn(A) = do(  -- ...
          -- j = 3
          a1 = ai[1],
          A2 = transpose((a1,a2)) ,
          a3 = ai[3] ,
          d3 = dot(A2p,a3) ,
          c3 = a3 -  ??? ,             -- (?1)
          test( c3 == (0,0,0) ,
                b3 = (1+dot(d3,d3))^(-1)*dot(d3,A2p) ,
                b3 = mpiV(c3)) ,
          B3 = A2p - ????,             -- (?2)
          A3p= zero(3,3) ,
               A3p[1] = B3[1] ,
               A3p[2] = B3[2] ,
               A3p[3] = b3    ,
          A3p   ) -- i.e. mpi

######################## END 3 x n ##################
```

- Which term is to be filled in at line (?1) at position `???` ?

- Which term is to be filled in at line (?2) at position `????` ?

b) Experiment with the matrices $A = ((1,1,1),(2,2,2),(3,3,5))$ and $A_{bad} = (((1,1,1),(1,1,0))$.

c) Run these commands:

```
" here we are .."
M = ((0,1/2,0),(0,1/2,0),(1,-1,0))
M
Mpi = Greville3xn( M )
Mpi
isMPI(Mpi,M)
```

Can you 'rescue' matrix A_{bad} in b) and calculate the *mip* A^{+}_{bad}?

d) Run these commands:

```
M22 = ((1,2,0),(1,2,0),(0,0,0))
M22
```

```
Mp22 = Greville3xn( M22 )
Mp22
isMPI(M22,Mp22)
```

Compare the result with 4.3.4.b.

e) Run these commands:

```
M = ((1,2))
M
M12 = ((1,2,0),(0,0,0),(0,0,0))
M12
Mp12 = Greville3xn( M12 )
Mp12
isMPI(M12,Mp12)
```

Compare the result with 4.3.2. Do you see a pattern?

In which respect are `Greville1xn` and `Greville2xn` superfluous? Why should we nevertheless keep them?

f) We have learned in e) that it is possible to 'reforest' a smaller typ matrix at his edges with zeros, so that it becomes e.g. a quadratic shape. After that one can calculate the Moore-Penrose-inverse of this new matrix and then get the Moore-Penrose-inverse of the original matrix by inspection of the printed result with the eyes.

Yet it is possible to peel out the wanted Moore-Penrose-inverse via matrix access commands.

Run these commands:

```
pM = (Mp12[1,1], Mp12[2,1])

qM=(0,0)
for(i,1,2, qM[i]=Mp12[i,1])
qM
isMPI(M,qM)
```

Compare with e). What do you recognize?

g) Use Wolfram's pseudoinverse widget in 4.2.2 to check the results in b) until e). Check *one* calculation by paper and pencil ♡.

Hint. ♡ Here is the solution to a): `???  = dot(A2,d3)` and `????  = outer(d3,b3)`.

4.4 The general Greville algorithm in Eigenmath.

```
#####    sequence of submatrices A1, A2, .. Ak i.e. A[:,1..k]
Ai(k) =  test( k=1, a[1],                -- case a[1] is vector
           do( AA=zero(k,dim(A,1)),      -- else k > 1
               for(i,1,k, AA[i]=a[i]),   -- (0)
               transpose(AA) ))

################## Greville m x n #################  A  typ m x n
--      procedure  GREVILLE  : A1+, A2+, .., An+  =  A+
####################################################  A+ typ n x m
Greville(A)=
     do( n=dim(A,2) ,
         m=dim(A,1) ,
         a = transpose(A) ,
         null = zero(2,m) ,
       do(  -- k = 1
         Ap = null ,
         Ap[1] = mpiV(a[1]) ,
            -- k = 2
         di = dot(Ap[1],a[2]) ,
         c  = a[2] - Ai(1)*di  ,
         test( c == null[1] ,
               b = (1+dot(di,di))^(-1)*dot(di,Ap[1]),
               b = mpiV(c)) ,
         B  = Ap[1] - di*b ,
         Ap = zero(2,m)  ,
               Ap[1] = B ,
               Ap[2] = b,
            -- k > 2
         do(for(k,3,n,
             di = dot(Ap,a[k]) ,
             c  = a[k] - dot(Ai(k-1),di),
             test( c == null[1],
                   b = (1+dot(di,di))^(-1)*dot(di,Ap),
                   b = mpiV(c) ),
             B = Ap - outer(di,b),
             -- print(B),                    -- (1)
             Ap = zero(k,m),
                  for(i,1,k-1, Ap[i] = B[i] ),
                  Ap[k] = b,
             Ap), -- close for, WATCH THE',' -- (2)
         Ap) -- close do around for-loop     -- (3)
        )   -- close inner do
       )  -- close outer do
################## END GREVILLE ###########################
```

Comment. We give some hints about the construction of the general function `Greville`() in EIGENMATH[36]. For readability we use index name k instead of j in the abstract formulation of the algorithm in 4.3.1. We write di instead of d, because d is a reserved identifier for differentation in EIGENMATH.
1. $a[k]$ denotes the k-th column of matrix A and Ap the future *mpi*-pseudoinverse of A.
2. The code lines for case $k = 1$ are known from 4.3.1 and are commented there. The code lines for $k = 2$ are known from 4.3.3 and 4.3.4 and case $k = 3$ is discussed in 4.3.5 and is in principe repeated here in the lines for $k > 3$.
ad (0): We use a helper function `Ai(k)`, which gives back the first k columns of matrix A, e.g. $Ai(2) = (a[1], a[2])$. This construction is sometimes denoted $A[., 1 : k]$ in other computer algebra languages.
ad (1): If you like you can sprinkle 'print' statements at interesting positions in the code to watch the output of intermediate results. You can turn on/off this feature by commenting on/off by writing/deleting the 2-fold comment hyphens '`--`' of EIGENMATH.
ad (2): Notice: The 'for'-loop is boxed in a 'do'-compound statement, *because 'for' does not return a value like 'do' does*![37] The ',' after '..`Ap)`' finalize the do-command, giving back the current result of `Ap` to be handled further in the next loop.
ad (3): here `Ap` is returned as value for the whole function call.

Exercise.
a. Load the helper functions `isMPI(P,A)` and `mpiV(A)` in your running EIGENMATH session. Then test the implementation of function `Greville()` e.g. via

```
Greville( ((1,2),(2,3)) )   -- n = m = 2

A=((1,1,1,3), (2,2,2,2), (3,3,3,5))
A
Api=Greville( A )           -- ..pi = (p)seudo(i)nverse
Api
isMPI(Api,A)
```

b. Compute the MOORE-PENROSE-inverse of $M = ((0, 1/2), (0, 1/2), (1, -1)$.
Check the result using Wolfram's widget.

♡

In this way we can solve any problem involving pseudoinverses satisfactorily and elegantly using the generalized MOORE-PENROSE-inverse *mpi* using function `Greville`. At this point, we end up our way through the linear algebra of pseudoinverses and their applications.

♡

[36]An implementation with *Octave* is in [4, p. 28] and with *Mathematica* e.g. in [16, p. 12-13]
[37]I thank George WEIGT for this hint.

4.5 Problems.

Before you try the following problems: Be sure that your toolbox *mpiBox.txt* contains the EIGENMATH-functions `isPinv`, `isMPI`, `isSolvable`, `solSet`, `mpiN`, `Ai()`, `Greville`. Load these functions on your iMac with the command `run("Downloads/mpiBox.txt")` into your actual session.

4.5.1 Determine a pseudoinverse.

Given is the matrix $A = ((1, 2, 3), (3, 4, 3), (6, 8, 6))$.
a. Determine a pseudoinverse of A using Elementary matrices $Em()$ to reach a final RREF.
b. Determine the MOORE-PENROSE-inverse of A.

4.5.2 Determine the MOORE-PENROSE-inverse.

Here is matrix $B = ((0, 1, 2), (0, 0, -2), (0, 2, 4))$.
b. Determine the MOORE-PENROSE-inverse of B.
b. Check the result with `isMPI`.

4.5.3 2x2 Linear Systems.

$$\begin{bmatrix} 2\cdot x + 3\cdot y = 1 \\ -x + 2\cdot y = 2 \end{bmatrix}, \begin{bmatrix} 2\cdot x + 3\cdot y = 1 \\ -x - 1.5\cdot y = 1 \end{bmatrix}, \begin{bmatrix} 2\cdot x + 3\cdot y = 1 \\ -x - 1.5\cdot y = 2 \end{bmatrix}$$

a. Check the solvability of the LS by calling `isSolvable`.
b. If necessary, calculate a particular solution `mpi(A)*B` or `pinv(A)*B`.
c. Determine the solution set using `solSet`(.) .
d. Give the solution set calculated in c. in parametric representation.
e. Try to visualize the solution set with e.g. *gnuPlot*.

4.5.4 Underdetermined 2x2 LSs.

$$\begin{bmatrix} 1\cdot x + 2\cdot y + 3\cdot z = 4 \\ 3\cdot x + 2\cdot y + 1\cdot z = 8 \end{bmatrix}, \begin{bmatrix} x + y + 2\cdot z = 5 \\ -2\cdot x - 2\cdot y - 4\cdot z = 11 \end{bmatrix}$$

a. Check the solvability of the systems.
b. Determine the solution set.
c. Give the solution set in parametric representation.
e. Visualize the solution set with paper and pencil.

4.5.5 Solve a linear 2x3 system of equations.

A Computer Algebra System (CAS) receives the following assignment:

```
    solve ({1*x + 2*y + 3*z = 4, 3*x + 4*y + 3*z = 2}, [x, y, z])
```

a. Help that CAS using EIGENMATH.
b. Solve this LS with different pseudo inverses.

c. Compare the solution sets. Justify the equivalence of the representations.

4.5.6 Solve another linear 2x3 system of equations.

The same Computer Algebra System (CAS) returns for the following request

```
    solve ({1*x  + 2*z = -1, -2*z=6, 2*y + 4*z = -2}, [x, y, z])
```

the answer `{ }` .

a. Study the solution set of this LS using pseudo inverses.

b. Study the solution set using the MOORE-PENROSE-inverseof the system matrix.

4.5.7 Calculate a pinv.

Compute a pseudo inverse (pinv) of the matrix

a. $((0, 1, 2, -1), (0, 0, -2, 6), (0, 2, 4, -2))$

b. $((0, 1, 2, -1, 2), (0, 0, -3, 5, -4), (0, 2, 1, 3, 0))$

4.5.8 A 3x3-LS.

Given is the Linear System

$$\begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 5 \end{bmatrix}$$

a. Check the solvability of the LS.

b. Calculate a particular solution and verify the solution property.

c. Determine the solution set.

d. Determine the solution set in parametric representation.

e. Try to visualize the solution set with e.g. *gnuPlot*.

4.5.9 An underdetermined 3x3 LS.

$$\begin{bmatrix} 1\cdot x + 2\cdot y + 3\cdot z = 4 \\ 1\cdot x + 2\cdot y + 3\cdot z = 4 \\ 1\cdot x + 2\cdot y + 3\cdot z = 4 \end{bmatrix}$$

a. Check the solvability of the LS. b. Calculate a particular solution.

c. Determine the full solution set.

d. Give the solution set calculated in parametric representation.

e. Try to visualize the solution set with e.g. *gnuPlot*.

4.5.10 Overdetermined 3x2 LS.

$$\begin{bmatrix} 2\cdot x + 3\cdot y = 1 \\ -x + 2\cdot y = 2 \\ x + 2\cdot y = 3 \end{bmatrix}, \begin{bmatrix} 1\cdot x + 2\cdot y = 3 \\ -x - 2\cdot y = -3 \\ 4\cdot x + 8\cdot y = 6 \end{bmatrix}$$

a. Check the solvability of the LS.

b. Calculate a best fit particular solution.

Mini	**Lexicon**
D	*E*
Beispiel	*example*
unterbestimmt	*underdetermined*
Gleichung	*equation*
Unbestimmte	*unknown*
Lösung	*solution*

4.5.11 Achilles' examples.

Check all examples in [1] using EIGENMATH.
Do not be bothered by the few German words. Use e.g. Google translate, if necessary.

4.5.12 Furlan's examples.

Do the examples 3 .. 9 in *The Yellow Book* [5] using EIGENMATH.
Do not use the $\substack{\text{D: Singular-wert-zerlegung} \\ \text{E: singular value decomposition (svd)}}$, use the `mpiBox.txt` instead.

4.5.13 Picaronny's example.

Do example 5 in [15] using EIGENMATH.
Look also at the compact formulation of the *Greville* algorithm on p. 4.

4.5.14 Test matrices of Tasić et al.

Do examples 4.1 and 4.2 in [16] using EIGENMATH. Compare the computation times of EIGENMATH vs. CAS *Mathematica* using the table in example 4.3.

4.5.15 Woermann's example.

Do the example about Linear Regression in [28] using EIGENMATH.
Translate the CAS Maxima code into EIGENMATH code and verify the calculation.

4.5.16 Wikipedia example.

Check the examples in [29] using EIGENMATH.

4.5.17 Petković example.

Try to compute the *mpi* for the matrices A, M and N of example 5.1 in [30] using EIGENMATH.

4.5.18 Potpourri I: Labus's examples.

Do $\substack{\text{D: Beispiel}\\\text{E: example}}$ 1.10, 1.13, 1.18 and 1.19 in [31] using EIGENMATH. MiniLexicon: $\substack{\text{D: Gleichung}\\\text{E: equation}}$.

4.5.19 YoTube lesson's.

Enjoy some of the video lessons [24], [25], [26] , [27] or [32].
Solve or verify the presented problems using EIGENMATH.

4.5.20 Ernst's example.

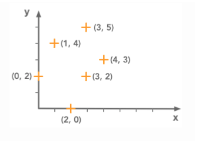
Try to solve or reproduce the problems on p.223 and p. 240 in [33]. There are many more aspects to do reading this script ...

4.5.21 Two examples at University of Stuttgart.

Do the two examples at the end of the text in [34] using EIGENMATH.

4.5.22 Potpourri II: Hadrien's examples.

Do the four examples of HADRIEN [35] using EIGENMATH instead of *numPy*.



We want to fit a line to this set of data points

What is in your opinion the pros and cons of both systems?

4.5.23 MacAusland's example.

Do it: p.9 of [36] using EIGENMATH. Check your answer using [37].

4.5.24 Example on MathePlanet.

Do the calculation of the pseudoinverse after Definition 2 of [38] using EIGENMATH.
Check your answer using an alternative software.

4.5.25 Caspary's test matrices for the Greville algorithm.

Abstract. An implementation of the Greville algorithm on a Motorola DSP96002 is presented. This algorithm enables us to calculate the pseudo-inverse of a matrix or the inverse of a regular matrix [1] [2]. An application to Least-Squares (LS) problems shows the relevant results obtained on a DSP96002 with a lower numerical complexity compared to other algorithms.

a. Do the calculation of the MOORE-PENROSE-inverse of matrix A in III in [39].
b. Do the application problem in IV.
c. Compare the MATLAB implementation of *Greville* with our implementation in EIGENMATH. There is also a flow diagram of the algorithm.

⋈

In conclusion, we have automated the discussion of linear systems of equations (LS) just as the discussion of functions in analysis. With the help of the pseudo-inverse of a matrix A, we can completely overlook and effectively determine the variety of solutions of any linear system of equations.

With that impression we want to finish our short excursion into the Linear Algebra with pseudoinverses and their applications. Once again:

♡

```
"Mathematics is
 not formulas,
or computations,
 or even proof,
  but IDEAS."

              Gilbert Strang, MIT/USA
```

♡

References

[1] Achilles, K. (2020): *Linksinverse Rechtsinverse Pseudoinverse Matrix.* url: `https://www.k-achilles.de/algorithmen/Matrix-Linksinverse-Rechtsinverse-Pseudoinverse.pdf`

[2] Blyth, T. S. & Robertson, E. F. (1998): *Basic Linear Algebra.* London: Springer.

[3] Borne, P. & Rotella, F. (1995): *Theorie et Pratique du Calcul Matriciel.* Paris: Editions Technip.

[4] Courrieu, P. (2005): Fast Computation of Moore-Penrose Inverse Matrices. arXiv.org: `https://arxiv.org/pdf/0804.4809.pdf`

[5] Furlan, P. (1995): Das Gelbe Rechenbuch. url: `http://www.das-gelbe-rechenbuch.de/download/Swz.pdf`

[6] Greville,T. N. E. (1960): "Some Applications of the Pseudoinverse of a Matrix." In: *SIAM Review*, 2 (1), 15-22. url: `http://www.jstor.org/stable/2028054`

[7] Hill, R. J. & Keagy, T. A. (1995): *Elementary Linear Algebra with DERIVE.* Bromley: Chartwell-Bratt.

[8] Köcher, M. (1983): *Lineare Algebra und analytische Geometrie.* Berlin: Springer.

[9] Lay, D. (^{2}1999): *Linear Algebra and its Applications.* Reading: Addison-Wesley.

[10] Lindner, W. (1999): "Pseudoinverse zur Lösung von linearen Gleichungssystemen. Ein Unterrichtskonzept realisiert mit DERIVE." In: *MNU* Vol. 52 (6), S. 341 - 340. ISSN 0025-5866

[11] Lindner, W. (2003): "CAS-supported Multiple Representations in Elementary Linear Algebra - The Case of the Gaussian Algorithm." In: *ZDM* Vol. 35 (2), S. 36 - 42.

[12] Lindner, W. (^{2}2004): *Ausgleichsrechnung, überbestimmte LGS und Pseudoinverse mit* MuPAD. Mathematik 1 x anders - Materialien und Werkzeuge für computerunterstütztes Lernen, Band 7. Paderborn: SicFace Software.

[13] Macdonald, A. (^{3}2010): *Linear and Geometric Algebra.* Amazon: Marston Gate. ISBN 9 781453 854938

[14] Möller, H. (1997): *Algorithmische Lineare Algebra.* Braunschweig: Vieweg.

[15] Picaronny, C. (2007): *Pseudo Inverse.* url: `http://nicolas.thiery.name/Enseignement/Agregation/Textes/PseudoInverseMatrice.pdf`

[16] Tasiíc, M. B.& Stanimirović, P. S. & Pepić, S. H. (2011): *About the generalized LM-inverse and the Weighted Moore-Penrose inverse.* arxiv.org: `https://arxiv.org/pdf/1104.1698.pdf`

[17] Schmidt, K. & Trenkler, G. (1998): *Moderne Matrix-Algebra.* Berlin: Springer.

[18] Seroul, R. (2000): *Programming for Mathematicans.* Berlin: Springer.

[19] Strang, G. (1998): *Introduction to Linear Algebra.* Wellesley: Wellesley-Cambridge Press.

[20] Szabo, F. (2002): *Linear Algebra - An Introduction using Maple.* Burlington: Harcourt/Academic Press.

[21] Weigt, G. (2020): Eigenmath *online Demo.*
url: `https://georgeweigt.github.io/eigenmath-demo.html`

[22] Williams, G (^{3}1996): *Linear Algebra with Applications.* Englewood: Morton.

[23] `https://en.m.wikipedia.org/wiki/System_of_linear_equations`

[24] `https://m.youtube.com/watch?v=EeY5a5who_s`

[25] `https://m.youtube.com/watch?v=vGowBXcur1k`

[26] `https://m.youtube.com/watch?v=09EV2e97oyA`

[27] `https://m.youtube.com/watch?v=5bxsxM2UTb4`

[28] `http://www.math.uni-bonn.de/people/woermann/MoorePenrose.pdf`

[29] `https://en.m.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse`

[30] `https://www.sciencedirect.com/science/article/pii/S0898122107006475`

[31] `http://www.mathematik.uni-kassel.de/~labus/Labus_Pseudoinverse2019.pdf`

[32] `https://www.helsinki.fi/en/unitube/video/6837fddf-af4c-47a9-9fba-c779a8c1b03b`

[33] `https://www.tu-chemnitz.de/mathematik/numa/lehre/numerik-2015/Folien/numerik5.pdf`

[34] `https://vhm.mathematik.uni-stuttgart.de/Vorlesungen/Lineare_Algebra/Folien_Pseudo-Inverse.pdf`

[35] `https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.9-The-Moore-Penrose-Pseudoinverse/`

[36] `http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-macausland-pseudo-inverse.pdf`

[37] `https://atozmath.com/MatrixEv.aspx?q=pseudoinverse&q1=4%2C0%3B3%2C-5%60pseudoinverse%60&dm=D&dp=8&do=1#PrevPart`

[38] `https://matheplanet.com/matheplanet/nuke/html/article.php?sid=742&ref=https://www.google.de/&f=1&ref=https://www.google.de&ff=y&rd3=1`

[39] `https://www.researchgate.net/publication/305565296_Implementation_of_the_Greville_algorithm_on_a_Motorola_DSP96002_Application_to_Least-Squares_problems`

$\bowtie$

Links checked 27.11.2020, wL

Dr. Wolfgang Lindner
Leichlingen, Germany
dr.w.g.Lindner@gmail.com
2020