

## Chapter 9

# Affine Functions: Local Analysis

Input-Output Pairs, 263 – Normalized Input-Output Rule, 266 – Local I-O Rule Near  $\infty$ , 268 – Local Graph Near  $\infty$ , 270 – Types of Local Graphs Near  $\infty$ , 271 – Local Features Near  $\infty$ , 272 – Localization At  $x_0$ , 273 – Local I-O Rule Near  $x_0$ , 276 – Local Coefficients Near  $x_0$ , 278 – Taylor Functions, 279 – Local Graph Near  $x_0$ , 280 – Local Features Near  $x_0$ , 281.

**Affine functions** are functions that are specified by an *input-output rule* that is of the form

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

where  $AFFINE_{a,b}$  is the *name of the function*<sup>1</sup> and where  $a$  and  $b$  stand for the two *bounded* numbers that are needed to *specify* the *affine function*  $AFFINE_{a,b}$ .

**1.** In order to facilitate our investigation of *affine functions*, it is necessary to begin by introducing some language special to *affine functions*. Given the affine function  $AFFINE_{a,b}$ , that is the function specified by the global input-output rule

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

---

<sup>1</sup>Educologists seem to have remained completely unaware that, some fifty years ago, “linear functions” ceased, albeit perhaps regrettably, to be “linear” in the sense of LINEAR MATHEMATICS, e.g. LINEAR ALGEBRA, and thus must now be called “affine functions”.

linear coefficient  
 linear term  
 constant coefficient  
 constant term  
 complete global  
 input-output rule  
 simplified global  
 input-output rule  
 loss of information  
 rule side

- The given signed number  $a$  is called the **linear coefficient** of the affine function  $AFFINE_{a,b}$  and  $ax^{+1}$ , that is the *linear coefficient* multiplied by one copy of the input,  $ax$ , is called the **linear term** of the affine function  $AFFINE_{a,b}$ .
- The given signed number  $b$  is called the **constant coefficient** of the affine function  $AFFINE_{a,b}$  and  $bx^0$ , the *constant coefficient* multiplied by no copy of the input,  $b$ , is called the **constant term** of the affine function  $AFFINE_{a,b}$ <sup>2</sup>.

**EXAMPLE 1.** The *affine function*  $NINA_{(-3,+5)}$  is the function whose input-output rule is

$$x \xrightarrow{NINA_{(-3,+5)}} NINA_{-3,+5}(x) = (-3)x^{+1} + (+5) \\ = -3x + 5$$

that is of the function whose *output* is equal to the *linear coefficient*  $-3$  multiplied by *one* copy of the input  $x$  to which is added the *constant term*  $+5$ .

- $-3x$  is the *linear term*,
- $+5$  is the *constant term*.

**2.** Unless there is a need for the above way of writing things, as there will be when, for instance, we have to deal with more than one affine function at a time, we shall only write whatever information is needed:

**a.** Instead of writing the **complete global input-output rule** as above,

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax^{+1} + bx^0$$

we may just write the **simplified global input-output rule**

$$x \xrightarrow{AFFINE_{a,b}} ax + b$$

since there is no **loss of information** in that the names of the coefficients still appear on the **rule side** of the global input-output rule, that is the phrase to the right of the  $=$  sign which *specifies* the output.

**EXAMPLE 2.** For instance, instead of writing

$$x \xrightarrow{NINA_{-3,+5}} NINA_{-3,+5}(x) = NINA_{-3,+5}(x) = (-3)x^{+1} + (+5)$$

we may just write the simplified global input-output rule

$$x \xrightarrow{NINA} -3x + 5$$

---

<sup>2</sup>Educologists will of course deplore this “hair splitting” but, had we not distinguished the *constant term* from the *constant coefficient*, they would have been the first to deplore the resulting confusion. At least one hopes so.

**b.** Actually, since the name of the function,  $AFFINE$ , occurs in two places, above the arrow,  $\xrightarrow{AFFINE}$ , and in the name of the output,  $AFFINE(x)$ , the *simplified global input-output rule* could be simplified even further and written into either one of two even simpler *ultra-simplified global input-output rules*. Indeed:

- Some people write the name of the function,  $AFFINE$ , only once, in the name of the output,  $AFFINE(x)$  and do not write the name of the function,  $AFFINE$ , on top of the arrow,  $\xrightarrow{AFFINE}$ .

**EXAMPLE 3.** Instead of writing

$$x \xrightarrow{MAMA} MAMA(x) = -2x + 4$$

some people just write

$$x \longrightarrow MAMA(x) = -2x + 4$$

because there still is no loss of information.

- Other people prefer to do the opposite and write the name of the function,  $AFFINE$ , only on top of the arrow,  $\xrightarrow{AFFINE}$  but not in the output.

**EXAMPLE 4.** Instead of writing

$$x \longrightarrow MAMA(x) = -2x + 4$$

other people prefer to write

$$x \xrightarrow{MAMA} -2x + 4$$

because there still is no loss of information.

In this text, though, we will use the *simplified global input-output rule* mentioned above, that is we will not write the coefficients in the name of the functions but we will write the name of the function both above the arrow and in the name of the output as:

- it cannot hurt,
- it is not much additional writing and
- it might prevent mistakes

**c.** On the other hand, instead of specifying an affine function by its *complete global input-output rule*

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

or its *simplified global input-output rule*

$$x \xrightarrow{AFFINE_{a,b}} ax + b$$

full name  
constant part

we may use just its **full name**, that is its name including the coefficients

$$AFFINE_{a,b}$$

because, again, there is no loss of information.

**EXAMPLE 5.** Instead of saying:

$$\begin{array}{l} \text{Given the function whose input-output rule is} \\ x \xrightarrow{NINA_{-3,+5}} NINA_{-3,+5}(x) = -3x + 5 \end{array}$$

or instead of saying

$$\begin{array}{l} \text{Given the function whose input-output rule is} \\ x \xrightarrow{NINA} -3x + 5 \end{array}$$

we can just as well say, without loss of information,

$$\text{Given the } \textit{affine} \text{ function } NINA_{-3,+5}$$

**3.** There are three ways to look at *affine functions* and which view we will take will depend on which is the most convenient for us in the mathematical situation we are in.

**a.** We can look at the affine function  $AFFINE_{a,b}$  as a combination of the first two *non-negative-exponent power functions*, that is of the two *exceptional power functions*.

$$\begin{array}{l} x \xrightarrow{IDENTITY} IDENTITY(x) = x^{+1} \\ x \xrightarrow{UNIT} UNIT(x) = x^0 \end{array}$$

that is as

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax^{+1} + bx^0$$

This is how we will look at polynomial functions in general.

**b.** We can look at the affine function  $AFFINE_{a,b}$  as the result of adding to the *constant function*

$$x \xrightarrow{CONSTANT_b} CONSTANT_b(x) = b$$

to be called the **constant part** of the affine function, an  $a$ -dilation of the *identity function*

$$x \xrightarrow{IDENTITY} IDENTITY(x) = x$$

that is the function

linear term

$$x \xrightarrow{a \cdot \text{IDENTITY}} a \cdot \text{IDENTITY}(x) = ax$$

to be called the **linear term** of the affine function. We will use this view when investigating and discussing *local graphs*.

**c.** We can look at the affine function  $\text{AFFINE}_{a,b}$  as the result of adding to the *linear function*, that is an  $a$ -dilation of the *identity function*,

$$x \xrightarrow{a \cdot \text{IDENTITY}} a \cdot \text{IDENTITY}(x) = ax$$

a *constant function*

$$x \xrightarrow{\text{CONSTANT}_b} \text{CONSTANT}_b(x) = b$$

We will use this view when investigating and discussing the *essential global graph*.

## POINTWISE ANALYSIS

### 9.1 Input-Output Pairs

As with any function, given the *input-output rule*, in order to get the output for a given input we must:

- i.** *Read and write* what the input-output rule says,
- ii.** Replace  $x$  in the input-output rule by the given input,
- iii.** Identify the resulting *specifying phrase*.

**1.** In the case of an affine function  $\text{AFFINE}_{a,b}$ , that is of a function whose global input-output rule is

$$x \xrightarrow{\text{AFFINE}_{a,b}} \text{AFFINE}_{a,b}(x) = ax + b$$

and given an input  $x_0$ , in order to get the output, we proceed as follows.

**i.** We *read and write* what the input-output rule says:

- The input-output rule *reads*:

“The output of  $\text{AFFINE}_{a,b}$  is obtained by *multiplying*  $a$  by 1 copy of the input and adding  $b$ ”

- We write

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

- ii. We indicate that  $x$  is about to be replaced by the given input  $x_0$

$$x \Big|_{x \leftarrow x_0} \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) \Big|_{x \leftarrow x_0} = ax + b \Big|_{x \leftarrow x_0}$$

which gives us the following *specifying-phrase*

$$= ax_0 + b$$

- iii. When we have *specific* values for  $a$ ,  $b$ , and  $x_0$ , we can then *identify* the *specifying-phrase*  $ax_0 + b$  to get the *specific* value of the output  $AFFINE_{a,b}(x_0)$ .

**EXAMPLE 6.** Given the input-output rule

$$x \xrightarrow{ALDA} ALDA(x) = -32.67x + 71.07$$

and given the input  $-3$ , in order to get the input-output pair, we proceed as follows.

- i. We *read* and *write* what the input-output rule says:

- The input-output rule *reads*:

“The output of *ALDA* is obtained by *multiplying*  $-32.67$  by 1 copy of the input and adding  $71.07$ ”

- We *write*

$$x \xrightarrow{ALDA} ALDA(x) = -32.67x + 71.07$$

- ii. We indicate that  $x$  is about to be replaced by the given input  $-3$

$$x \Big|_{x \leftarrow -3} \xrightarrow{ALDA} ALDA(x) \Big|_{x \leftarrow -3} = -32.67x + 71.07 \Big|_{x \leftarrow -3}$$

which gives us the following *specifying-phrase*

$$= (-32.67) \cdot (-3) + 71.07$$

iii. We identify the *specifying-phrase* to get the specific output.

$$\begin{aligned} &= +98.01 + 71.07 \\ &= +169.08 \end{aligned}$$

**2.** In practice we usually will *not* write all of this since there is a lot of redundant information.

i. On the input side, instead of writing

$$x \Big|_{x \leftarrow x_0}$$

we will write just the *result* of replacing  $x$  by  $x_0$ , that is

$$x_0$$

ii. On the output side, instead of writing the full

$$AFFINE_{a,b}(x) \Big|_{x \leftarrow x_0}$$

we will similarly write just the *result* of replacing  $x$  by  $x_0$ , that is

$$AFFINE_{a,b}(x_0)$$

iii. But, on the “rule side” will we keep the indication that  $x$  is to be replaced by  $x_0$

$$= ax + b \Big|_{x \leftarrow x_0}$$

because that is where the work will be.

So, altogether, given an input  $x_0$ , we will write

$$x_0 \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x_0) = ax + b \Big|_{x \leftarrow x_0}$$

**EXAMPLE 7.**

$\begin{aligned} -3 &\xrightarrow{ALDA} ALDA(-3) = -32.67x + 71.07 \Big _{x \leftarrow -3} \\ &= (-32.67) \cdot (-3) + 71.07 \\ &= +98.01 + 71.07 \\ &= +169.08 \end{aligned}$
--

3. Altogether, and depending on the circumstances, we can then write:

$$x_0 \xrightarrow{AFFINE} ax_0 + b$$

or

$$AFFINE(x_0) = ax_0 + b$$

or

$(x_0, ax_0 + b)$  is an *input-output pair* for the function *AFFINE*

**EXAMPLE 8.** Altogether, and depending on the circumstances, we can then write

$$-3 \xrightarrow{ALDA} +169.08$$

or

$$ALDA(-3) = +169.08$$

or

$(-3, +169.08)$  is an *input-output pair* for the function *ALDA*

## 9.2 Normalized Input-Output Rule

Because, in the case of *affine functions*, we will always be able to carry out our investigations all the way to the *quantitative* stage, it may seem that there is not much need for anything else: just do it. Still, it generally does pay to have an overall view of whatever it is that we are doing and it is no less true in the case of *affine functions* because it will help us later see *affine functions* within the general scheme of “polynomial functions”.

1. In the case of *affine functions*, the features of the input-output rule that will matter to us will be:

- the *sign* of the *linear coefficient*
- the *sign* of the *constant coefficient*

In other words, we will carry out our investigations without concerning ourselves with the *size* of either the *linear coefficient* or the *constant coefficient*.

2. From the *qualitative* viewpoint, there will therefore be *four types* of *affine functions*:

Sign linear coefficient	Sign constant coefficient
+	+
	-
-	+
	-

**EXAMPLE 9.** The function *BETA* whose input-output rule is

normalize

$$x \xrightarrow{BETA} BETA(x) = -16.93x + 46.03$$

is an affine function whose *input-output rule* has the following *features*

- The *linear coefficient* is *Negative*,
  - The *constant coefficient* is *Positive*.
3. So, given an affine function, we will **normalize** it as follows:
- We will *normalize the linear coefficient* as follows:

If the <i>linear coefficient</i> is:	we will <i>normalize</i> it to:
Positive	+1
Negative	-1

- We will *normalize the constant term* as follows:

If the <i>constant term</i> is:	we will <i>normalize</i> it to:
Positive	+1
Negative	-1

**EXAMPLE 10.** The function *CICA* whose *input-output rule* is

$$x \xrightarrow{CICA} CICA(x) = -13.20x + 24.48$$

will be *normalized* to

$$x \xrightarrow{CICA} CICA(x) = -1x + 1$$

which, however, we will usually write

$$= -x + 1$$

4. So, there are four *types* of *affine* functions with the following *normalized* input-output rules:

TYPE	Normalized input-output rule
<i>PosPos</i>	$x \xrightarrow{PosPos} PosPos(x) = +x + 1$
<i>PosNeg</i>	$x \xrightarrow{PosNeg} PosNeg(x) = +x - 1$
<i>NegPos</i>	$x \xrightarrow{NegPos} NegPos(x) = -x + 1$
<i>NegNeg</i>	$x \xrightarrow{NegNeg} NegNeg(x) = -x - 1$

principal term near  $\infty$

**NOTE.** As usual, “ $-x$ ” is best read as “opposite of  $x$ ”.

## LOCAL ANALYSIS NEAR INFINITY

### 9.3 Local Input-Output Rule Near Infinity

As with *all* functions, past, present and future, our first move in the investigation of *affine functions* will usually be to get the *local graph near  $\infty$* .

As will be the case with the other polynomial functions, given *any* affine function *AFFINE*, it will be convenient to begin by specifying a new function whose local graph near  $\infty$  will turn out to be the local graph near  $\infty$  of *AFFINE*.

1. Given an affine function  $AFFINE_{a,b}$  that is a function specified by the *global* input-output rule

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

the global input-output rule says:

- i. multiply the (signed) linear coefficient by one copy of the input, and then
- ii. add the (signed) constant coefficient.

We will now be using:

- The fact that a *bounded* number *multiplied* by a *large-in-size* number gives a result that is a *large-in-size* number,
- The fact that, regardless of its *sign*, adding a *bounded* number to a *large-in-size* number does *not* change the *size* of the result which is still *large-in-size*.

So, when the input is *large-in-size*, after we have multiplied the linear coefficient by a copy of the input the result is large-in-size and there is no point adding the *constant term*, regardless of its *sign*, since it is bounded and therefore does not change the *size* of the output.

In other words, in the case of *affine functions*, multiplying the linear coefficient by one copy of the input results in the **principal term near  $\infty$**  of the output, that is the linear term  $ax$  makes up “most of the output” when the input is near  $\infty$  and the *constant part* of the output,  $b$ , can be safely ignored when the input is near  $\infty$ .

2. So, given an affine function  $AFFINE_{a,b}$ , the new function that we will introduce,

[...] approximate local input-output rule near  $\infty$

*PRINCIPAL TERM of  $AFFINE_{a,b}$*

will be the function whose output is the *cubing term* of  $AFFINE_{a,b}$ , that is the function specified by the global input-output rule

$$x \xrightarrow{PPL.TERM.of\ AFFINE_{a,b}} PPL.TERM.of\ AFFINE_{a,b}(x) = ax$$

3. However, even for *large* inputs, the output of *PPL TERM of AFFINE* is not quite equal to the output of *AFFINE* because, even for *large* inputs, while the *constant part* of *AFFINE* is “too small to matter here”, it is not 0.

**EXAMPLE 11.** 0.33 is not equal to  $\frac{1}{3}$  because  $3 \times 0.33 = 0.99$  while  $3 \times \frac{1}{3} = 1$  and so we can only write  $\frac{1}{3} = 0.33 + [...]$

But, in mathematics, we do want to write *equalities* if only because they are easier to work with. So, in order to be able to write an *equality*, we will use [...] as a shorthand for “something too small to matter here”. We will then be able to write, completely truthfully,

$$x|_{x \text{ near } \infty} \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x)|_{x \text{ near } \infty} = ax + [...]$$

which we will of course call the **approximate local input-output rule near  $\infty$**  of the affine function  $AFFINE_{a,b}$

**NOTE.** This is the reason we used “global input-output rule” rather than just “input-output rule” —which must have looked like overdoing it—back at the beginning of the text. It was to allow us, starting right now, to emphasize the difference between *global* input-output rules which work for *any* input and a *local* input-output rules which work only for inputs *in a given neighborhood*.

**EXAMPLE 12.** Given the function  $ALMA_{-71.05,-41.83}$ , that is the affine function specified by the global input-output rule

$$x \xrightarrow{ALMA} ALMA(x) = -71.05x - 41.83$$

the function *PPL.TERM. of ALMA* is the function specified by the global input-output rule

$$x \xrightarrow{PPL.TERM\ of\ ALMA} PPL.TERM.\ of\ ALMA(x) = -71.05x$$

and we can write truthfully

$$x|_{x \text{ near } \infty} \xrightarrow{JOAN} ALMA(x)|_{x \text{ near } \infty} = -71.05x + [...]$$

In other words, we have:

procedure by reduction

**THEOREM 1 (Approximation Near  $\infty$ ).** *Near  $\infty$  the output of an affine function is approximately the same as the output of its principal term.*

## 9.4 Local Graphs Near Infinity

Since we already investigated the local graph of the *identity function*, the **Approximation Near  $\infty$  Theorem** then gives us a **procedure by reduction**, that is a procedure that reduces the current problem to a problem we have already been able to solve, for finding the local graph near  $\infty$  of *affine functions*:

- i. Get the *approximate local input-output rule near  $\infty$*
- ii. Normalize it,
- iii. Draw its *local graph near  $\infty$* .

**EXAMPLE 13.** Given the affine function  $ALIX_{-21.36, -45.78}$  that is the function specified by the *global input-output rule*,

$$x \xrightarrow{ALIX} ALIX(x) = -21.36x - 45.78$$

find its local graph near  $\infty$ .

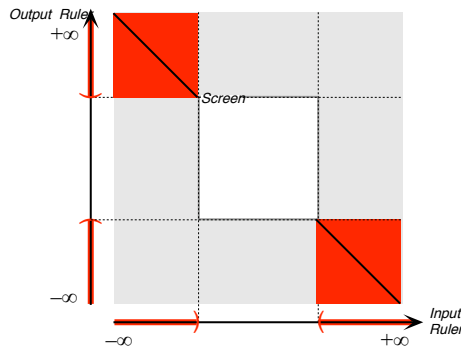
- i. The approximate *local input-output rule near  $\infty$*  is:

$$x|_{x \text{ near } \infty} \xrightarrow{ALIX} ALIX(x)|_{x \text{ near } \infty} = -21.36x^{+1} + [\dots]$$

- ii. We *normalize ALIX* to  $ALIX^*$

$$x|_{x \text{ near } \infty} \xrightarrow{ALIX^*} ALIX^*(x)|_{x \text{ near } \infty} = -x^{+1} + [\dots]$$

- iii. The *local graph near  $\infty$*  of  $ALIX^*$  is:



## 9.5 Types of Local Graphs Near Infinity

straight  
diagonal

Since the local graph near  $\infty$  of an affine function is the graph of the exponent +1 power function which is its *principal part*, we get that the local graph near  $\infty$  of an affine function does not depend on the constant term so that there are only two *types* of local graph near  $\infty$  depending on the *sign of the linear coefficient* namely

- the exponent +1 power function

$$x \xrightarrow{IDENTITY} IDENTITY(x) = (+1)x^{+1}$$

or

- the exponent +1 power function

$$x \xrightarrow{OPPOSITE} OPPOSITE(x) = (-1)x^{+1}$$

They are shown in the table below as seen from two viewpoints:

**i.** As seen from “not too far”, that is we see the screen and only the part of the local graph near  $\infty$  that is near the *transition*, that is for inputs that are *large* but not “that” large so that we can still see the *slope*.

**ii.** As seen from “faraway”. Indeed, in order to see really large inputs, we need to be “faraway” and the parts of the local graph near  $\infty$  that we see are still **straight** and **diagonal**.

Input-output rule	From “not too far”	From “faraway”
$x \xrightarrow{IDENTITY} IDENTITY(x) = (+1)x^{+1}$		
$x \xrightarrow{OPPOSITE} OPPOSITE(x) = (-1)x^{+1}$		

standard notation

## 9.6 Local Features Near Infinity

As we saw in **Chapter 3**, Section 7, we can read the local features near  $\infty$  off the local graph near  $\infty$ .

1. In order to state **Local Feature Theorems** as simply as possible, it will be convenient to use the **standard notation** in which we think of

$\swarrow$ and $\cup$ as being <i>positive</i> , that is as $+$ $\searrow$ and $\cap$ as being <i>negative</i> , that is as $-$
--

2. Then, based on the local graphs in the previous section, we have:

**THEOREM 2 (Local Features Near  $\infty$ ).** *Given an affine function  $AFFINE_{a,b}$ , the local features near  $\infty$  are:*

- Height-sign  $AFFINE|_{x \text{ near } \infty} = (\text{Sign } a, -\text{Sign } a,)$
- Slope-sign  $AFFINE|_{x \text{ near } \infty} = (\text{Sign } a, \text{Sign } a,)$
- *There is no concavity.*

**EXAMPLE 14.** Given the affine function  $NAOMI_{-40.38, -94.21}$ , that is the function specified by the global input-output rule

$$x \xrightarrow{NAOMI} NAOMI(x) = -40.38x - 94.21$$

we find its local features near  $\infty$  as follows.

i. The local features near  $\infty$  according to the **Local Features Theorem** are:

- Height-sign  $|_{x \text{ near } \infty} = (-, +)$
- Slope-sign  $|_{x \text{ near } \infty} = (-, -)$
- There is no concavity.

which, translated back into our language, gives

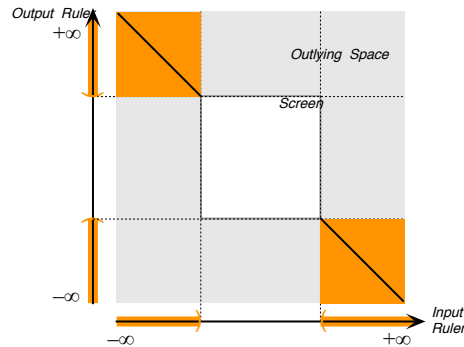
- Height-sign  $|_{x \text{ near } \infty} = (-, +)$
- Slope-sign  $|_{x \text{ near } \infty} = (\searrow, \searrow)$
- There is no concavity.

ii. Should we want to check that the **Local Features Theorem** gave us the correct information, we would get the normalized approximate local input-output rule near  $\infty$

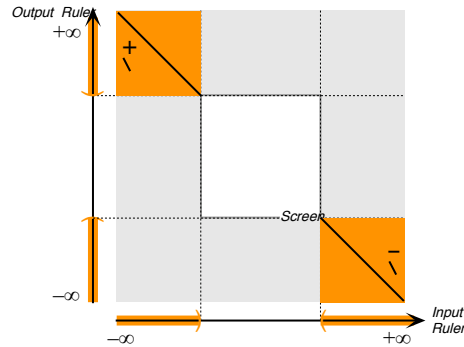
$$x|_{x \text{ near } \infty} \xrightarrow{NAOMI^*} NAOMI^*(x)|_{x \text{ near } \infty} = -x + [...]$$

and then the local graph near  $\infty$

characterize  
computationally



from which we get the following local features



which are the same as those that were given by the **Local Features Theorem** keeping in mind that the local graph near  $\infty$  is viewed here by **Mercator** rather than by **Magellan**.

## LOCAL ANALYSIS NEAR A FINITE INPUT

### 9.7 Localization At $x_0$

In the case of the *power functions* we investigated in **Chapters 5**, **Chapter 6** and **Chapter 7**, the only *finite input* near which we carried a local analysis was 0. From now on, though, and with any other function, we will need to be able to carry out a local analysis near *any* finite input  $x_0$ .

The only real difficulty, at least the only new thing we have to cope with, is the fact that we do not know how to **characterize computationally** inputs that are near a given finite input  $x_0$ .

1. Recall that, in the local analysis of *power functions* near 0, we had already run into the same difficulty inasmuch as we could not replace in the global input-output rule  $x$  by “near 0” and still be able to compute.

As we saw there, the way out was the fact that inputs that are near 0

are *characterized computationally* by the fact that they are *small* so that we were able to replace

$$x \text{ near } 0$$

by

$$\textit{small}$$

and then use the **Multiplication Theorem** to compute with *small*, that is using the fact that multiplying copies of *small* resulted in a number *smaller* than the original and therefore itself *small*.

**EXAMPLE 15.** Given the function *NYK* whose *input-output rule* is

$$x \xrightarrow{NYK} NYK(x) = +20x^{+5}$$

and given an input near 0, that is given a *small* input, we can compute the *output* because we can compute with *small*:

$$\begin{aligned} x \Big|_{x \leftarrow \textit{small}} \xrightarrow{NYK} NYK(x) \Big|_{x \leftarrow \textit{small}} &= (+20)x^{-5} \Big|_{x \leftarrow \textit{small}} \\ &= \frac{+20}{\underbrace{x \cdot \dots \cdot x}_{\textit{odd number of copies of } x}} \Big|_{x \leftarrow \textit{small}} \\ &= \frac{+20}{\underbrace{(-\textit{small}) \cdot \dots \cdot (-\textit{small})}_{\textit{odd number of copies of } -\textit{small}}} \\ &= \frac{+20}{-\textit{small}} \\ &= -\textit{large} \end{aligned}$$

**2.** When we want to investigate a function near a given finite input  $x_0$  instead of near 0, we encounter the same difficulty but we cannot use the words “input near  $x_0$ ” as replacement for  $x$  in the global input-output rule because we do *not* know how to *compute* with the words “input near  $x_0$ ”.

**EXAMPLE 16.** Given the function *ABE* whose *input-output rule* is

$$x \xrightarrow{ABE} ABE(x) = +3x - 17$$

and given an input near  $x_0$ , we try to replace  $x$  by “input near  $-4$ ”:

$$\begin{aligned} x \Big|_{x \leftarrow \text{input near } -4} \xrightarrow{NYK} ABE(x) \Big|_{x \leftarrow \text{input near } -4} &= +3x - 17 \Big|_{x \leftarrow \text{input near } -4} \\ &= +3 \cdot (\text{input near } -4) - 17 \end{aligned}$$

But, since we have no idea what “near  $-4$ ” amounts to, we cannot compute

local input  
 $h$

$$+3 \cdot (\text{input near } -4) - 17$$

and we are stuck.

**3.** The way out is to use the *location relative to  $x_0$*  of  $x$ , which we introduced at the very end of **Chapter 2**. In other words, we will use the **local input**

$$x - x_0 = u$$

**EXAMPLE 17.** Instead of saying:

$$3.14 \text{ is near } 3$$

we will write

$$3.14 = 3 + 0.14$$

The reason of course is that when  $x$  is near  $x_0$ , then  $u$  is going to be near 0 and so  $u$  will be *characterized computationally* by being *small*. In fact, when  $u$  is *small*, we will use the letter  $h$  instead to save us the trouble of having to say that “where  $u$  is small”.

Then, when  $x$  is near  $x_0$  we will be able to replace

$$x \text{ near } x_0$$

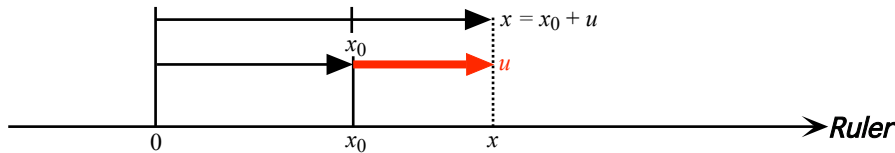
by

$$x_0 + h$$

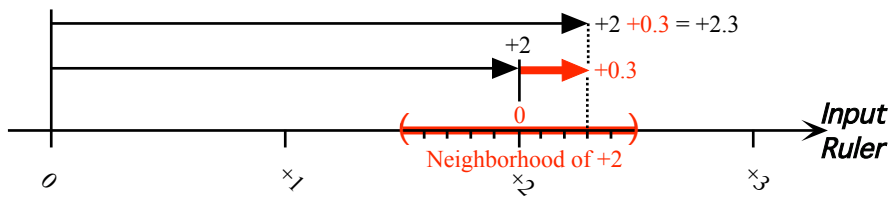
**EXAMPLE 18.** In order to investigate a function near the given a finite input  $+2$ , for instance the near finite input  $+2.3$ , we will use the location of 2.3 relative to the given finite input 2:

$$+2.3 - (+2) = +0.3$$

Graphically, we will have the following qualitative picture:



**EXAMPLE 19.** Given the finite input  $+2$ , then the location of the near finite input  $+2.3$  relative to  $+2$  is  $+0.3$ :



localize  
 order of diminishing sizes  
 local function

## 9.8 Local Input-Output Rule Near A Finite Input

When we investigated power functions near 0, we replaced  $x$  by *small* in the global input-output rule and *computed* the size of the output using the **Multiplication Theorem**. Here, we will proceed as follows

**i.** We will replace  $x$  by  $x_0 + h$  (where  $h$  is *small*) in the global input-output rule

**ii.** We will use the **Multiplication Theorem** to compute with  $h$

In other words, even though we will want to be able to investigate quadratic functions near *any* finite input  $x_0$ , we will proceed in pretty much exactly the same manner as we did with power functions.

**1.** More precisely, given an *affine* function  $AFFINE_{a,b}$ , that is a function specified by the global input-output rule

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

and given an input  $x_0$ , we find the local input-output rule of  $AFFINE$  near  $x_0$  pretty much as one would expect:

**i.** We **localize** the function  $AFFINE$  at the given input  $x_0$ , that is we count inputs from the given input  $x_0$  instead of from the origin of the input ruler. In other words, we use the *location* of  $x$  in relation to the given input  $x_0$ , that is we replace in the global input-output rule  $x$  by  $x_0 + h$ , (where  $h$  is *small*).

$$\begin{aligned} x|_{x \leftarrow x_0+h} &\xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x)|_{x \leftarrow x_0+h} = ax + b|_{x \leftarrow x_0+h} \\ &= a[x_0 + h] + b \\ &= ax_0 + ah + b \end{aligned}$$

**ii.** Collecting “like terms”,

$$= [ax_0 + b] + ah$$

where we **MUST** write the constant term *ahead* of the linear term because the linear term is *smaller-in-size* than the constant term and we always want to write the terms in **order of diminishing sizes**.

**2.** Since  $x_0$  is given for the duration of the local investigation and since it is therefore  $h$  that is the actual input, we will often think in terms of the **local function**  $AFFINE_{(x_0)}$ , that is the function which returns for  $h$  the same output that the global function  $AFFINE$  would return for  $x = x_0 + h$ .

The computation we just did gives us the *local input-output rule* that specifies the *local function*  $AFFINE_{(x_0)}$ :

**THEOREM 3 (Local Input-Output Rule).** *Given the affine function  $AFFINE$  specified by the global input-output rule*

$$x \xrightarrow{AFFINE} AFFINE(x) = ax + b$$

*the local function  $AFFINE_{(x_0)}$  is specified by the local input-output rule*

$$h \xrightarrow{AFFINE_{(x_0)}} AFFINE_{(x_0)}(h) = [ax_0 + b] + [a]h$$

**EXAMPLE 20.** Given the affine function  $MARA_{+3,+17}$ , that is given the function whose global input-output rule is

$$x \xrightarrow{MARA} MARA(x) = +3x + 17$$

and given the input,  $-5$ , we *localize* the function  $MARA$  at  $-5$  as follows:

$$\begin{aligned} x|_{x=-5+h} \xrightarrow{MARA} MARA(x)|_{x=-5+h} &= +3x + 17|_{x=-5+h} \\ &= +3(-5 + h) + 17 \\ &= -15 + 3h + 17 \end{aligned}$$

and, collecting terms of the same order of magnitude,

$$= +2 + 3h$$

The local rule of the function  $MARA$  at the given input  $-5$  is therefore the function whose *local input-output rule* is:

$$h \xrightarrow{MARA_{(-5)}} MARA_{(-5)}(h) = +2 + 3h$$

**3.** Thus the local rule of a given affine function near a given *finite input*  $x_0$  is very much in the same spirit as the local rule of the given function at  $\infty$ .

A difference, though, is that, in the case of the local rule at a finite input, the local input is counted from  $x_0$  while, in the case of the local rule at  $\infty$ , the input remains counted from 0 (because nobody has ever succeeded counting down from  $\infty$ ).

local constant coefficient  
local linear coefficient

## 9.9 Local Coefficients Near A Finite Input

When looking only for *one* of the local features, instead of computing the whole local input-output rule, we will only compute the single term of the local input-output rule that controls the local feature. At first, getting just this one single term rather than the whole local input-output rule will look more difficult because it cannot be done in a humdrum manner but that will not last.

More precisely, given the affine function  $AFFINE_{a,b}$ , that is a function specified by the global input-output rule

$$x \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) = ax + b$$

and whose local input-output rule is therefore

$$h \xrightarrow{AFFINE_{(x_0)}} AFFINE_{(x_0)}(h) = [ax_0 + b] + [a]h$$

we will say that:

- $[ax_0 + b]$  is the **local constant coefficient**,
- $[a]$  is the **local linear coefficient**,

and we will now investigate how to get just a single one of the local coefficients without getting the whole local input-output rule. At first, getting just this one single term rather than the whole local input-output rule will look more difficult than getting the whole local input-output rule but, with a little bit of practice, writing less and less each time, this will soon get easy.

We begin by writing

$$\begin{aligned} x \big|_{x \leftarrow x_0+h} \xrightarrow{AFFINE_{a,b}} AFFINE_{a,b}(x) \big|_{x \leftarrow x_0+h} &= ax + b \big|_{x \leftarrow x_0+h} \\ &= a(x_0 + h) + b \end{aligned}$$

**a.** To find the *local constant coefficient*, we proceed as follows:

- i.** The contribution of  $a(x_0 + h)$  to the *local constant term* will be  $a \cdot x_0$
- ii.** The contribution of  $b$  to the local constant term will be  $b$

Altogether, the *local constant term* adds up to

$$[ax_0 + b]$$

**b.** To find the *local linear coefficient*, we proceed as follows

- i.** The contribution of  $a(x_0 + h)$  to the *local linear term* will be  $a \cdot h$

ii. The contribution of  $b$  to the *local linear term* will be nothing. Altogether, the *local linear term* adds up to

1<sup>st</sup> Taylor  $AFFINE_{a,b,c,d}$

$$\left[ a \right] h$$

Admittedly, here, the economy is not exactly staggering. On the other hand, though, this will serve to get us used to being able to compute each local coefficient independently of the others since, as we continue with our investigation of polynomial functions, any other way will eventually become ultra complicated.

## 9.10 Taylor Functions

We saw just above that the *local constant coefficient* for an affine function  $AFFINE_{a,b}$  is:

$$ax_0 + b$$

It so happens, of course, that  $ax_0 + b$  is the output returned by  $AFFINE$  for the input  $x_0$ :

$$\begin{aligned} x_0 &\xrightarrow{AFFINE} AFFINE(x) \Big|_{x \leftarrow x_0} = ax + b \Big|_{x \leftarrow x_0} \\ &= ax_0 + b \end{aligned}$$

In order for the *local linear coefficient* also to be the output of a function, we will automatically associate with  $AFFINE$  another function whose purpose will indeed be, given an input  $x_0$ , to output the *local constant coefficient*.

More precisely, since the local input-output rule of  $AFFINE$  near  $x_0$  is

$$h \xrightarrow{AFFINE_{(x_0)}} AFFINE_{(x_0)}(h) = \left[ ax_0 + b \right] + \left[ a \right] h$$

we will say that:

- 1<sup>st</sup> **Taylor**  $AFFINE_{a,b,c,d}$  (read as “First Taylor of  $AFFINE$ ”) is the function that outputs the *local linear coefficient*, that is the coefficient of  $h^1$  in the local input-output rule of  $AFFINE$  near  $x_0$ .

In other words, the function 1<sup>st</sup> Taylor  $AFFINE_{a,b}$  is specified by the global input-output rule

$$x \xrightarrow{1^{\text{st}} \text{ Taylor } AFFINE} 1^{\text{st}} \text{ Taylor } AFFINE(x) = a$$

0<sup>th</sup> Taylor

In fact, we will think of the original function *AFFINE* as being its own 0<sup>th</sup> **Taylor** because it completes the pattern inasmuch as it outputs the coefficient of  $h^0$

$$x \xrightarrow{0^{\text{th}} \text{ Taylor } \textit{AFFINE}} 0^{\text{th}} \text{ Taylor } \textit{AFFINE}(x) = ax + b$$

This of course looks very much like using a sledge hammer to drive in a brad. However, this will extend very nicely to the other polynomial functions and will help us seeing the overall pattern. The real reason, though, is that, later on, in DIFFERENTIAL CALCULUS, and slightly modified for the purpose of making their calculation easier<sup>3</sup>, Taylor functions will appear as the absolutely and totally central concept.

## 9.11 Local Graph Near A Finite Input

The local graph of a given affine function  $\textit{AFFINE}_{a,b}$  near a given input  $x_0$  is the graph of the *local function*  $\textit{AFFINE}_{(x_0)}$ , that is the graph of the function specified by the *local input-output rule* of  $\textit{AFFINE}_{a,b}$  near  $x_0$ .

So, in order to get the local graph of the given affine function  $\textit{AFFINE}_{a,b}$  near the given input  $x_0$ , we:

- i. Get the local graph of the *constant term* of the local rule  $\textit{AFFINE}_{x_0}$ ,
- ii. Get the local graph of the *linear term* of the local rule  $\textit{AFFINE}_{x_0}$ ,
- iii. “Add” the local graph of the *linear term* to the local graph of the *constant term* as we did at the very end of **Chapter 7**.

**EXAMPLE 21.** Given the affine function *MARA* whose local rule near  $-5$ ,  $\textit{MARA}_{(-5)}$  is given by the local input-output rule

$$h \xrightarrow{\textit{MARA}_{(-5)}} \textit{MARA}_{(-5)} = +2 + 3h$$

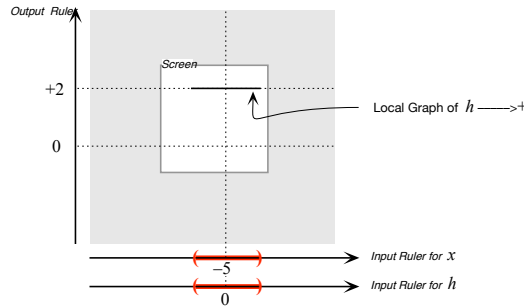
find the local graph near  $-5$

- i. We get the local graph of the *constant term* of the local rule  $\textit{MARA}_{(-5)}$ :

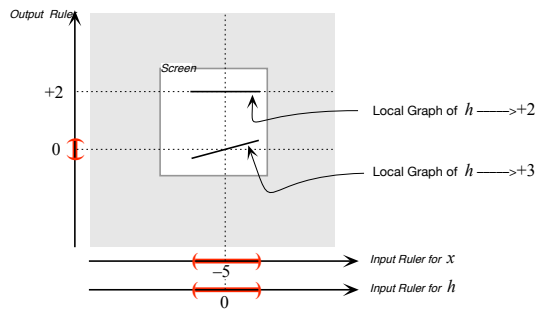
---

<sup>3</sup>Here, Educologists are of either one of two persuasions. One is that, quite obviously, Taylor functions are much too difficult a concept to have a place in “precalculus”. The other asserts, just as forcefully, that, just as obviously, anything less than the *derivative functions* is bound to “confuse the students”. However, the two camps agree on this being a non-negotiable issue and will brook no discussion. The point, though, is that Taylor functions differ from derivative functions only by a factorial whose only purpose is to make the computation of derivatives *recursive* which, here, seems rather beside the point. But then, Educologists are not wont to be affected by reasons mathematical—or otherwise.

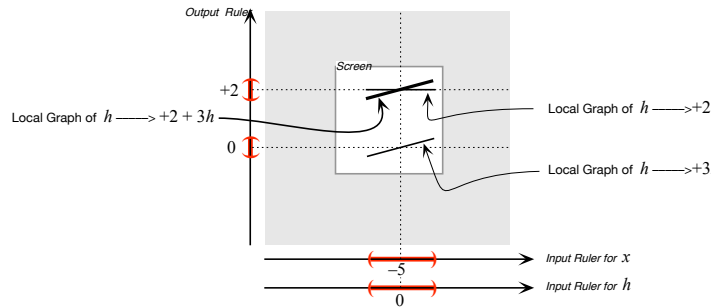
control



ii. We get the local graph near  $x_0$  of the *linear term* of the local rule  $MARA_{(-5)}$ ,



iii. We “add” the local graph of the *linear term* of the local rule  $MARA_{(-5)}$  to the local graph of the *constant term* of  $MARA_{(-5)}$ .



## 9.12 Local Features Near A Finite Input

Once we have the local graph of the function *AFFINE* near  $x_0$ , we can “read” the local features off the local graph as discussed in **Chapter 4**. This is indeed a good way to proceed initially because it is “visual”.

However, the local graph acts only as an intermediary between the local input-output rule and the local features and it is really the *coefficients* in the local input-output rule that **control** the *local features* so that, if all we want is only one particular local feature, it is very inefficient to have to compute

total local height

the whole local input-output rule to get the local graph. Indeed,

- Why compute coefficients that control features we don't need?
- As we investigate more functions, computing the whole local input-output rule is rapidly turning out to be more and more labor-intensive, if only because the addition formulas get to be longer and longer and more and more complicated but for other reasons as well, as we will see with *rational functions*.

More precisely, given the affine function  $AFFINE_{a,b}$ , that is a function specified by the global input-output rule

$$x \xrightarrow{AFFINE} AFFINE(x) = ax + b$$

what *controls* the local features of  $AFFINE$  near  $x_0$  is the the local input-output rule near  $x_0$

$$h \xrightarrow{AFFINE(x_0)} AFFINE_{(x_0)}(h) = [ax_0 + b] + [a]h$$

and therefore it is the *local coefficients* that each control a particular local feature.

Since the local coefficients are the output of the Taylor functions of  $AFFINE$ , we will state the resulting theorems in terms of the Taylor functions of  $AFFINE$  so as to display the overall pattern.

1. The *local constant term*

$$[ax_0 + b]$$

is what controls the *local height*. This is because, as long as  $h$  remains *small*, that is as long as  $x$  remains near  $x_0$ , the other term of the local input-output rule, namely the *linear term* contributes very little *height* to the **total local height** and certainly not enough to *change the sign* of the *total local height*. The *local constant coefficient* is therefore what determines the *height-sign* near any finite input  $x_0$  which will therefore be determined by the sign of  $AFFINE(x_0)$ . In other words, by the sign of 0<sup>th</sup> Taylor  $AFFINE(x_0)$ :

**THEOREM 4 (Height-sign Near  $x_0$ ).** *For any affine function  $AFFINE_{a,b}$ :*

- When  $AFFINE(x_0) = +$ , Height-sign  $AFFINE|_{\text{near } x_0} = (+, +)$
- When  $AFFINE(x_0) = -$ , Height-sign  $AFFINE|_{\text{near } x_0} = (-, -)$
- When  $AFFINE(x_0) = 0$ , Height-sign  $AFFINE|_{\text{near } x_0}$  is given by

the sign of 1<sup>st</sup> Taylor  $AFFINE(x_0)$

In other words:

- When  $0^{th}$  Taylor  $AFFINE(x_0) = +$ , Height-sign  $AFFINE|_{near\ x_0} = (+, +)$
- When  $0^{th}$  Taylor  $AFFINE(x_0) = -$ , Height-sign  $AFFINE|_{near\ x_0} = (-, -)$
- When  $0^{th}$  Taylor  $AFFINE(x_0) = 0$ , Height-sign  $AFFINE|_{near\ x_0}$  is given by the sign of 1<sup>st</sup> Taylor  $AFFINE(x_0)$

2. The local linear term

$$\left[ a \right] h$$

is what controls the *local slope*. This is because the *constant term* has no slope.

The local *linear coefficient* is therefore what determines the *slope-sign* near any finite input  $x_0$  which will be determined by the sign of 1<sup>st</sup> Taylor  $AFFINE(x_0)$ :

**THEOREM 5 (Slope-sign Near  $x_0$ ).** For any quadratic function  $AFFINE_{a,b}$ :

- When  $1^{st}$  Taylor  $AFFINE(x_0) = +$ , Slope-sign  $AFFINE|_{near\ x_0} = (\swarrow, \swarrow)$
- When  $1^{st}$  Taylor  $AFFINE(x_0) = -$ , Slope-sign  $AFFINE|_{near\ x_0} = (\searrow, \searrow)$
- $1^{st}$  Taylor  $AFFINE(x_0)$  cannot be equal to 0

Using the *standard notation*, that is thinking of

$\swarrow$ and $\cup$ as being <i>positive</i> , that is as $+$ $\searrow$ and $\cap$ as being <i>negative</i> , that is as $-$
--

allows for a much nicer statement of the theorem, one that brings out the similarity of the **Slope-sign Near  $x_0$  Theorem** with the **Height-sign Near  $x_0$  Theorem**:

**THEOREM 6 (Slope-sign Near  $x_0$ )** For any affine function  $AFFINE_{a,b}$ :

- When  $1^{st}$  Taylor  $AFFINE(x_0) = +$ , Slope-sign  $AFFINE|_{near\ x_0} = (+, +)$
- When  $1^{st}$  Taylor  $AFFINE(x_0) = -$ , Slope-sign  $AFFINE|_{near\ x_0} = (-, -)$
- $1^{st}$  Taylor  $AFFINE(x_0)$  cannot be equal to 0