Oscillatori Virtuali Caotici: Sintesi del segnale digitale mediante l'uso di sistemi caotici

Luca Spanedda

March 10, 2022

Abstract

I segnali caotici nella sintesi digitale possono essere utili per implementare oscillatori e/o segnali di controllo tempo-varianti. Un problema dei sintetizzatori digitali e degli effetti audio, è che sono spesso caratterizzati da suoni molto lontani da quelli prodotti nel mondo fisico, a causa della natura precisa e tempo-invariante della loro generazione del segnale nel mondo digitale.
Nel computer, i dettagli che in natura nel suono si verificano normalmente in modo imprevedibile, devono essere accuratamente sequenziati fino al punto di esaurimento delle risorse del computer e del programmatore.
Adottare segnali caotici nella sintesi e nel controllo, può essere dunque un modo per produrre suoni più naturali di quelli generati attraverso le tecniche più standard di sintesi digitale, con metodi computazionali più economici.
In questo studio andremo ad implementare dei circuiti nel linguaggio di programmazione Faust(GRAME) per rappresentare discretamente alcuni modelli caotici, ed alcuni stocastici, che possano essere utili nella generazione di texture sonore di sintesi. Estrapoleremo da questi codici alcune topologie per rappresentare i circuiti corrispondenti alle equazioni differenziali implementate, ed andremo ad utilizzare il linguaggio di programmazione Python per ottenere alcuni plot che rappresentino il comportamento di questi modelli.

I. Introduzione alla teoria del caos

≪Può il battito delle ali di una farfalla in Brasile
scatenare un tornado in Texas ?≫
(Lorenz all'American Association for the Advancement of Sciences, 1979)

La teoria del caos postula che esista una classe di fenomeni naturali che possano essere modellati da sistemi deterministici non lineari.
Quando si parla di caos ci si riferisce ad un certo tipo di comportamento di un sistema dinamico: determinato dalla sensibilità alle condizioni iniziali, imprevedibilità a lungo termine, orbite periodiche dense. In altri termini un sistema caotico amplifica le piccole differenze: porta i fenomeni microscopici a un livello macroscopico. E' dunque nell'amplificazione di queste piccole deviazioni delle condizioni iniziali che si annida il caso.
Nonostante siano perfettamente individuate le equazioni differenziali che descrivono questi sistemi: al variare delle condizioni iniziali varia il comportamento del sistema stesso negli stadi successivi a quello iniziale.
L'ipotesi che sistemi deterministici possano sviluppare comportamenti impredicibili fu teorizzata per la prima volta dal matematico francese Henri Poincaré già nello studio del problema dei tre corpi (1890).
Tuttavia la nascita vera e propria di questa teoria scientifica si verifica però nel 1963, quando Edward Norton Lorenz pubblica il suo articolo Deterministic Nonperiodic Flow, nel quale tratta del comportamento caotico in un sistema semplice e deterministico, con la formazione di un attrattore strano introducendo il medesimo concetto.
I sistemi caotici, proprio come quelli stocastici, appartengono alla classe dei sistemi dinamici e dei sistemi complessi. Solo che mentre nei sistemi caotici i comportamenti sono deterministici ed è possibile prevedere il risultato se si conoscono le condizioni iniziali del sistema ed il suo comportamento; dunque a parità delle condizioni iniziali il risultato è prevedibile.
Nei modelli stocastici il comportamento dipende da delle probabilità, che possono essere calcolate tramite il calcolo delle probabilità. Non è possibile dunque determinare a parità delle condizioni iniziali il proprio comportamento.

II. Sistemi Dinamici e Complessi

Un sistema dinamico è un modello matematico che rappresenta un oggetto (sistema), con un numero finito di gradi di libertà che evolve nel tempo secondo una legge deterministica;
Mentre per sistema complesso si intende un sistema dinamico a multicomponenti, ovvero composto da diversi sottosistemi che tipicamente interagiscono tra loro.
Tipicamente un sistema dinamico viene rappresentato analiticamente da un'equazione differenziale, espressa poi in vari formalismi, e identificato da un vettore nello spazio delle fasi: lo spazio degli stati del sistema, dove "stato" è un termine che indica l'insieme delle grandezze fisiche, dette variabili di stato, i cui valori effettivi "descrivono" il sistema in un certo istante temporale.
Si possono identificare due tipologie di sistema dinamico:

- Dinamico Discreto
- Dinamico Continuo

In questi modelli: un sistema è Dinamico Discreto se l'evoluzione avviene ad intervalli discreti di tempo, e Dinamico Continuo se l'evoluzione è continua e definita da un'equazione differenziale.

Ci sono sistemi che non variano col passare del tempo, mentre da altri ci si aspetta un effettivo cambiamento, e sono detti rispettivamente:

- Sistemi tempo-invarianti.
- Sistemi tempo-varianti.

III. Non Linearità

Ciò che accomuna un sistema complesso ad un sistema caotico è la non linearità. In questa visione di complessità i sistemi caotici sono considerati un sottoinsieme dei sistemi complessi: la complessità si manifesta infatti sulla soglia della caoticità. Mentre nel mondo fisico la non-linearità è insita alla natura delle cose, nel mondo digitale, queste non-linearità devono essere accuratamente programmate ed introdotte all'interno degli algoritmi, per raggiungere la complessità. Di pro, si può ottenere con un certo controllo il comportamento inverso: si può arrivare a dei comportamenti Lineari a partire da modelli complessi. Ad esempio si può pensare alla costrizione delle Equazioni differenziate Caotiche, per guidarne e modificarne il loro comportamento in base alle proprie esigenze e scopi. vedi Dario Sanfilippo in Bibliografia:
[D.Sanfilippo - Constrained Differential Equations as Complex Sound Generators. Proceedings of the 18thSound and Music Computing Conference, June 29th{ July 1st2021]

IV. Equazioni differenziali

Un'equazione differenziale è un'equazione che lega una funzione incognita alle sue derivate. La funzione derivata di una funzione rappresenta il tasso di cambiamento di una funzione rispetto a una variabile, vale a dire quindi la crescita (o decrescita) che avrebbe una funzione in uno specifico punto spostandosi di pochissimo dal punto considerato.
Può essere anche descritta come un set di equazioni relative al range di cambiamento di un numero sconosciuto e la sua derivata.
ad esempio:

$$y = sin(y), \tag{1}$$

Esistono dei modelli di equazioni differenziali a comportamenti caotici. I segnali caotici possono infatti essere generati trovando soluzioni numeriche a determinate equazioni differenziali, o mediante l'uso iterativo di mappe di primo ritorno.
Alcune delle equazioni differenziali più utilizzate in grado di generare segnali caotici sono:
Lotka-Volterra, Van der Pol, Lorenz, Rössler, Hindmarsh-Rose, and Thomas.
Ma ovviamente esiste una grande varietà di modelli e implemetazioni possibili col fine di ottenere comportamenti caotici.
Nel nostro caso per la generazione del caos andremo a cercare delle soluzioni discrete ad alcune equazioni differenziali caotiche, ed in seconda istanza a modificarne una con il fine di arrivare ad una mia personale implementazione per generare texture sonore.

V. Discretizzazione ed implementazione delle Equazioni differenziali

In FAUST (Grame) i codici che vengono scritti possono essere anche compilati in Topologie/Schemi a blocchi di circuiti, oltre che in applicazioni audio di vario tipo. Questo ci permette nel nostro caso di poter esplorare le equazioni differenziali, nelle loro forme discrete, esplose in topologie che ne illustrino il comportamento ed il funzionamento. Partiamo dall'illustrazione di una semplice equazione differenziale che introduce il concetto di iterazione di una funzione:

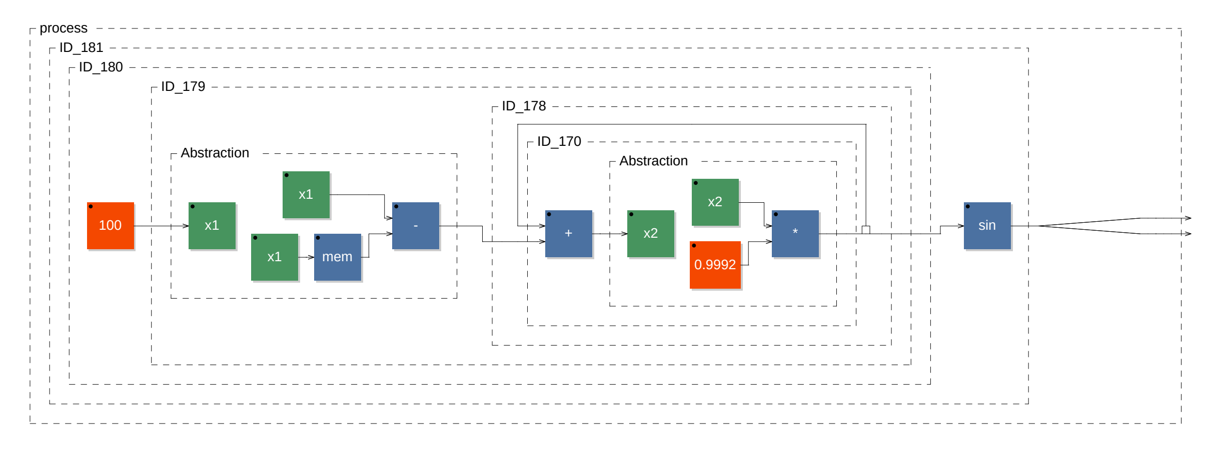


Figure 0.1: Schema di ricorsione in Faust

In questo caso il sistema non è caotico, poiché determinate le condizioni iniziali si può determinare di conseguenza un preciso comportamento per ogni istante di tempo successivo al primo (ad ogni iterazione).
Mandato un Impulso di Dirac (discreto) di valore 100 e della durata di un campione in ingresso al sistema, il valore passa per la moltiplicazione 0.9992, e l'uscita della moltiplicazione è rimandata in ingresso al sistema stesso al campione successivo, in questo caso fino al suo azzeramento.
Sfruttando il comportamento della rampa che viene generata per guidare la funzione seno, si ottiene una modulazione di frequenza direttamente proporzionale alla velocità in cui decade la rampa, che genera un timbro simile a quello di un colpo su una grancassa. Andiamo a vedere il codice in Faust:

```
---------------------------------------------------------------------------
Codice Faust: differential_equation_circuit.dsp
---------------------------------------------------------------------------
// import Standard Faust library
// https://github.com/grame-cncm/faustlibraries/
import("stdfaust.lib");

// basic circuit for the iteration of a function :
// for solve differential equations (without chaotic behaviors)

recursion(x,a) = output
with{
   Dirac(x) = x-(x:mem);
   differentialequation(x) = x*a;
   circuit = x : Dirac : (+ : differentialequation)~ _;
   output = sin(circuit);
   };

process = recursion(100, 0.9992) <: _,_;
```

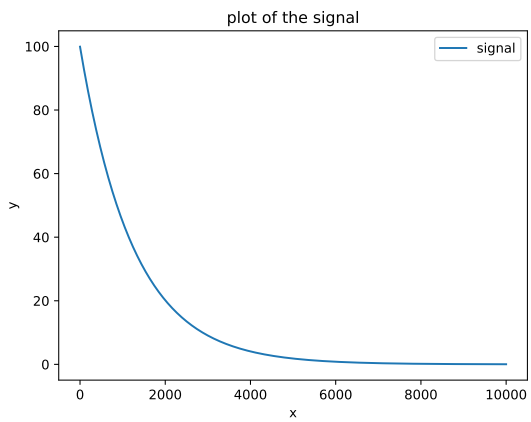
Ora che abbiamo chiarito la rappresentazione di un circuito in codice Faust, scriviamo lo stesso codice anche in Python; la libreria mathplotlib ci permette di esportare dei plot grafici per visualizzare il comportamento della funzione iterata. Ho deciso di creare uno script in python, che ha il compito di creare un plot grafico a partire dai valori di ogni iterazione in uscita dalla funzione e salvati in un .txt. Un plot esporta i valori della rampa dall'interno della funzione seno, mentre uno semplicemente i valori di ogni iterazione della funzione.

```
--------------------------------------------------------------------------------
Codice Python: recursionsin.py & recursionline.py
--------------------------------------------------------------------------------
# ITERAZIONE DI UNA FUNZIONE

# ISTRUZIONI PER LA COMPILAZIONE
# Per la compilazione in un file .txt eseguire in bash il comando:
# $ sudo python3 file.py >> file.txt
# (dove file è il nome del file da compilare)

import math

# PROCESSO:
# la funzione process(a) è il processo deterministico
# su cui si vuole fare - feedback - reiterazione dell'output
# in return avviene l'operazione
def process(a):
   return a*0.9992

# ITERAZIONI:
# è definito qui quante iterazioni ripete il processo di caos deterministico:
iterazioni = int(10000)

# VALORE DI PARTENZA:
# valore di partenza del processo caos deterministico:
valore= float(100)

for j in range (0,iterazioni):
   # mando il valore nel process
   out=process(valore)
                # - math.sin - rimuovere o lasciare per:
                # recursionsin.py & recursionline.py
   valori = print(math.sin(out))
   # feedback: - prendo l'uscita da out e la reimmetto nel process
   valore=out
```

Andiamo a vedere invece lo script in python, che ha il compito di compilare i valori in uscita dalla funzione in un plot grafico utilizzando le librerie mathplotlib e numpy.

```
--------------------------------------------------------------------------------
Script Python: datagraphplot.py
--------------------------------------------------------------------------------
# PYTHON SCRIPT FOR PLOT A DATA GRAPH
# plot a serie of data numers written in a list like:
# 1
# 2
# 3
# ...

import matplotlib.pyplot as plt
import numpy as np

# External inputs from terminal
filename = input("Enter your data file name (add .txt extension): ")
plotname = input("Enter the name for your plot (add .png/.pdf/... extension): ")

x = np.loadtxt(filename)
plt.plot(x, label='signal')

plt.xlabel('x')
plt.ylabel('y')
plt.title('plot of the signal')
plt.legend()
plt.savefig(plotname)
```

A seguito i plot ottenuti dai codici, ed una tabella contenente invece i primi valori nel file di testo generato da recursionline.py dove è possibile leggere i valori generati da ogni iterazione della funzione.



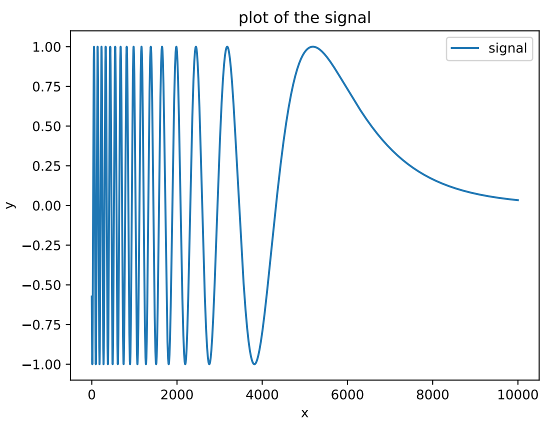


Figure 0.2: Plot grafico dell'iterazione della funzione

Figure 0.3: Plot grafico dell'iterazione in seno

```
--------------------------------------------------------------------------------
Primi valori da: recursionlineplot.txt
--------------------------------------------------------------------------------

99.92 99.840064 99.7601919488 99.68038379524096 99.60063948820476 99.5209589766142 99.4413422094329
99.36178913566536 99.28229970435683 99.20287386459334 99.12351156550167 99.04421275624927 98.96497738604427
98.88580540413543 98.80669675981211 98.72765140240426 98.64866928128234 98.56975034585732 98.49089454558063
98.41210182994416 98.33337214848021 98.25470545076142 98.17610168640081 98.09756080505169 98.01908275640764
97.94066749020251 97.86231495621035 97.78402510424539 97.705797884162 97.62763324585467 97.54953113925798
97.47149151434657 97.39351432113509 97.31559950967818 97.23774703007044 97.15995683244638 97.08222886698043
97.00456308388684 96.92695943341973 96.849417865873 96.77193833158029 96.69452078091503 96.6171651642903
96.53987143215886 96.46263953501314 96.38546942338513 96.30836104784642 96.23131435900814 96.15432930752092
96.0774058440749 96.00054391939965 95.92374348426412 95.8470044894767 95.77032688588513 95.69371062437642
95.61715565587691 95.5406619313522 95.46422940180712 95.38785801828567 95.31154773187104 95.23529849368555
95.15911025489059 95.08298296668667 95.00691658031332 94.93091104704907 94.85496631821142 94.77908234515685
94.70325907928073 94.6274964720173 94.55179447483968 94.47615303925981 94.4005721168284 94.32505165913493
94.24959161780762 94.17419194451338 94.09885259095776 94.023573508885 93.94835465007789 93.87319596635783
93.79809740958474 93.72305893165708 93.64808048451175 93.57316202012414 93.49830349050804 93.42350484771563
93.34876604383746 93.27408703100238 93.19946776137758 93.12490818716847 93.05040826061874 92.97596793401024
92.90158715966304 92.8272658899353 92.75300407722335 92.67880167396157 92.6046586326224 92.5305749057163
92.45655044579173 92.38258520543509 92.30867913727074 92.23483219396093 92.16104432820576 92.0873154927432
92.01364564034901 91.94003472383673 91.86648269605766 91.7929895099008 91.71955511829289 91.64617947419825
91.57286253061889 91.4996042405944 91.42640455720192 91.35326343355615 91.2801808228093 91.20715667815105
91.13419095280852 91.06128360004628 90.98843457316624 90.91564382550771 90.8429113104473 90.77023698139895
90.69762079181383 90.62506269518038 90.55256264502422 90.4801205949082 90.40773649843227 90.33541030923352
90.26314198098613 90.19093146740134 90.11877872222742 90.04668369924964 89.97464635229024 89.9026666352084
89.83074450190024 89.75887990629872 89.68707280237368 89.61532314413178 89.54363088561647 89.47199598090798
89.40041838412324 89.32889804941594 89.2574349309764 89.18602898303162 89.1146801598452 89.04338841571732
88.97215370498473 88.90097598202074 88.82985520123512 88.75879131707413 88.68778428402047 88.61683405659325
88.54594058934798 88.4751038368765 88.40432375380699 88.33360029480394 88.2629334145681 88.19232306783644
88.12176920938217 88.05127179401465 87.98083077657944 87.91044611195817 87.84011775506859 87.76984566086453
87.69962978433584 87.62947008050837 87.55936650444396 87.4893190112404 87.4193275560314 87.34939209398658
87.27951258031139 87.20968897024714 87.13992121907094 87.07020928209569 87.00055311467001 86.93095267217828
86.86140791004054 86.7919187837125 86.72248524868553 86.65310726048658 86.5837847746782 86.51451774685844
86.44530613266096 86.37614988775483 86.30704896784462 86.23800332867035 86.16901292600741 86.1000777156666
86.03119765349406 85.96237269537126 85.89360279721495 85.82488791497718 85.7562280046452 85.68762302224148
85.61907292382368 85.55057766548462 85.48213720335224 85.41375149358956 85.34542049239468 85.27714415600076
85.20892244067596 85.14075530272342 85.07264269848125 85.00458458432246 84.936580916655 84.86863165192167
84.80073674660014 84.73289615720286 84.6651098402771 84.59737775240487 84.52969985020295 84.46207609032278
84.39450642945052 84.32699082430695 84.2595292316475 84.19212160826218 84.12476791097558 84.0574680966468
83.99022212216948 83.92302994447175 83.85589152051617 83.78880680729975 83.7217757618539 83.65479834124442
83.58787450257142 83.52100420296937 83.454187399607 83.38742404968731 83.32071411044755 83.25405753915919
83.18745429312786 83.12090432969335 83.05440760622959 82.98796408014461 82.92157370888049 82.85523644991338
82.78895226075345 82.72272109894485 82.65654292206568 82.59041768772803 82.52434535357786 82.458325877295
82.39235921659316 82.3264453292199 82.26058417295651 82.19477570561814 82.12901988505365 82.0633166691456
81.99766601581028 81.93206788299763 81.86652222869122 81.80102901090827 81.73558818769953 81.67019971714937
81.60486355737565 81.53957966652975 81.47434800279652 81.40916852439427 81.34404118957475 81.2789659566231
81.2139427838578 81.14897162963071 81.084052452327 81.01918521036514 80.95436986219684 80.88960636630708
80.82489468121403 80.76023476546906 80.69562657765668 80.63107007639455 80.56656522033343 80.50211196815717
80.43771027858264 80.37336011035977 80.30906142227148 80.24481417313366 80.18061832179515 80.11647382713771
80.052380648076 79.98833874355753 79.92434807256268 79.86040859410463 79.79652026722934 79.73268305101556
79.66889690457474 79.60516178705107 79.54147765762143 79.47784447549533 79.41426219991493 79.350730790155
79.28725020552287 79.22382040535845 79.16044134903416 ...
```

VI. La Mappa Logistica

Per introdurre delle equazioni differenziali che presentano comportamenti caotici, un primo passo può essere quello di implementare dei sistemi caotici ad un equazione, l'equazione della mappa logistica è uno di questi. La mappa logistica è una mappa polinomiale di secondo grado, introdotta dal biologo Robert May nel 1976. May elaborò la mappa a partire dall'equazione logistica del matematico Pierre François Verhulst che, tra il 1838 e il 1847, la utilizzò per descrivere l'evoluzione di una popolazione nel tempo.
L'equazione differenziale che descrive la mappa logistica è la seguente:

$$x_n + 1 = rx_n(1 - xn), \tag{2}$$

Dove xn è un numero compreso tra 0 e 1, e rappresenta il rapporto tra la popolazione esistente e quella massima possibile in un anno n, e quindi x0 rappresenta il rapporto tra la popolazione iniziale (all'anno 0) e quella massima;
r è un numero positivo e rappresenta il tasso combinato tra la riproduzione e la mortalità.
Andiamo ora ad osservare la topologia della mappa logistica nello schema a blocchi:

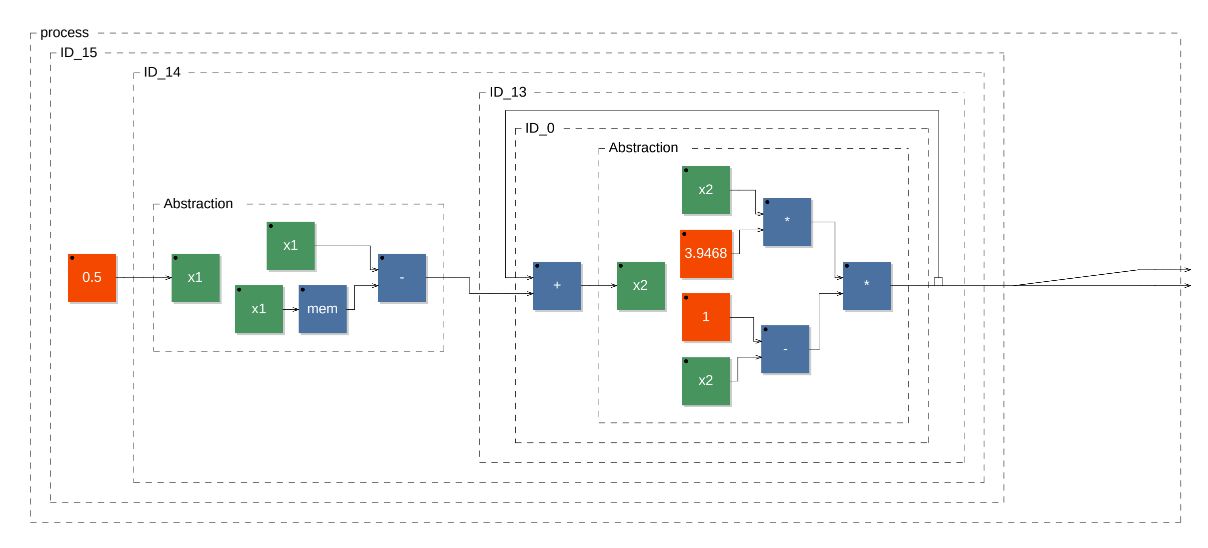
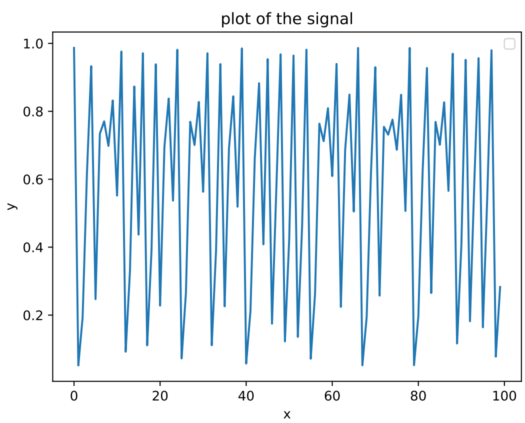


Figure 0.4: Schema Topologico della Mappa Logistica

In questa rappresentazione i valori dati sono di x = 0.5 e di r =3.9468. Come per l'implementazione dell'equazione precedente (a questa della mappa logistica), determinate le condizioni iniziali di x ed r, si può determinare di conseguenza un preciso comportamento per ogni istante di tempo successivo al primo, ma in questo caso il sistema è caotico, poichè al variare di queste configurazioni ne variano gli esiti. Andiamo a rappresentare il corrispettivo codice appartenente allo schema in Faust ed in Python, ed il corrispettivo plot del codice Python:

```
--------------------------------------------------------------------------------
Codice Faust: logisticmap.dsp
--------------------------------------------------------------------------------
// import Standard Faust library
// https://github.com/grame-cncm/faustlibraries/
import("stdfaust.lib");

// Logistic Map - simple non-linear dynamical equation
// starting population (x) = number from 0. to 1.
// resources of the population (r) = number from 0. to 4.

logisticmap(x,r) = mapfunc
with{
   Dirac(x) = x-(x:mem);
   lmap(x) = x*r*(1-x);
   mapfunc = x : Dirac : (+ : lmap)~ _;
   };

process = logisticmap(0.5,3.9468) <: _,_;
```

```
------------------------------------------------------------------------------------
Codice Python: logisticmap.py
------------------------------------------------------------------------------------
# MAPPA LOGISTICA

# equazione differenziale della mappa logistica
def logisticmap(x, r):
    return x * r * (1 - x)

# anni = iterazioni del processo (cicli iterativi)
iterazioni = int(100)

# VALORI DI PARTENZA:
# valori di x ed r
x = float(0.5)
r = float(3.9468)


# loop degli anni, e passa i valori aggiornati nuovamente
# all'interno dell'equazione differenziale
for j in range (0,iterazioni):
    # mando il valore nell'equazione differenziale
    out = logisticmap(x,r)
    # print dei valori in uscita
    print(out)
    # feedback:
    x=out
```



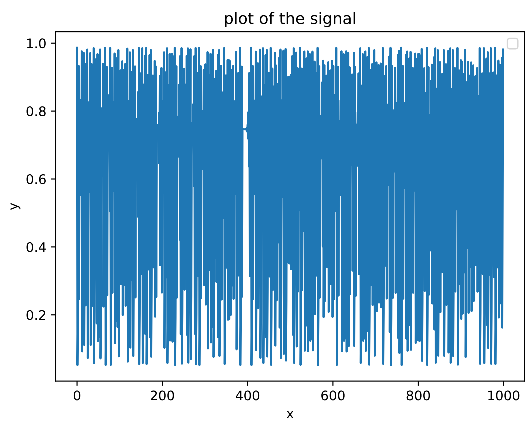


Figure 0.5: Plot mappa logistica per 100 anni

Figure 0.6: Plot mappa logistica per 1000 anni

VII. Attrattore di Lorenz

Spostando ora la nostra attenzione, verso dei sistemi con dei comportamenti caotici descritti da più equazioni differenziali del primo ordine, la nostra attenzione può ricadere sull'attrattore di Lorenz. L'attrattore di Lorenz fu il primo esempio di un sistema di equazioni differenziali a bassa dimensionalità in grado di generare un comportamento caotico.
Venne scoperto da Edward N. Lorenz, del Massachusetts Institute of Technology, nel 1963.
Semplificando le equazioni del moto alle derivate parziali che descrivono il movimento termico di convezione di un fluido, Lorenz ottenne un sistema di tre equazioni differenziali del primo ordine. La versione continua dell'equazione differenziale che descrive il modello di Lorenz è la seguente:

$$x = a(y - x), y = -xz + rx - y, z = xy - bz, \tag{3}$$

Dove la variabile x è proporzionale all'ampiezza della circolazione della velocità del fluido nell'anello del fluido, il positivo rappresenta il moto in senso orario ed il negativo il senso antiorario. La variabile y è la differenza di temperatura tra i fluidi su e giù, e z è la distorsione dalla linearità del profilo della temperatura verticale.
Andiamo ora ad osservare la topologia dell'Attrattore di Lorenz discretizzato nello schema a blocchi:

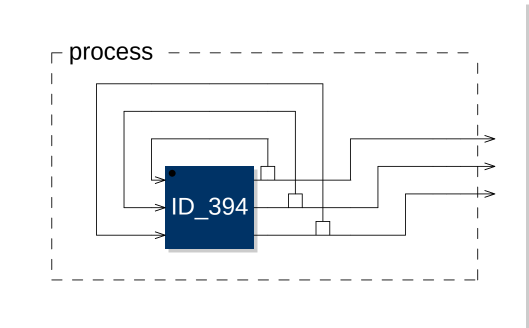


Figure 0.7: Schema di Reiterazione delle equazioni del sistema di Lorenz

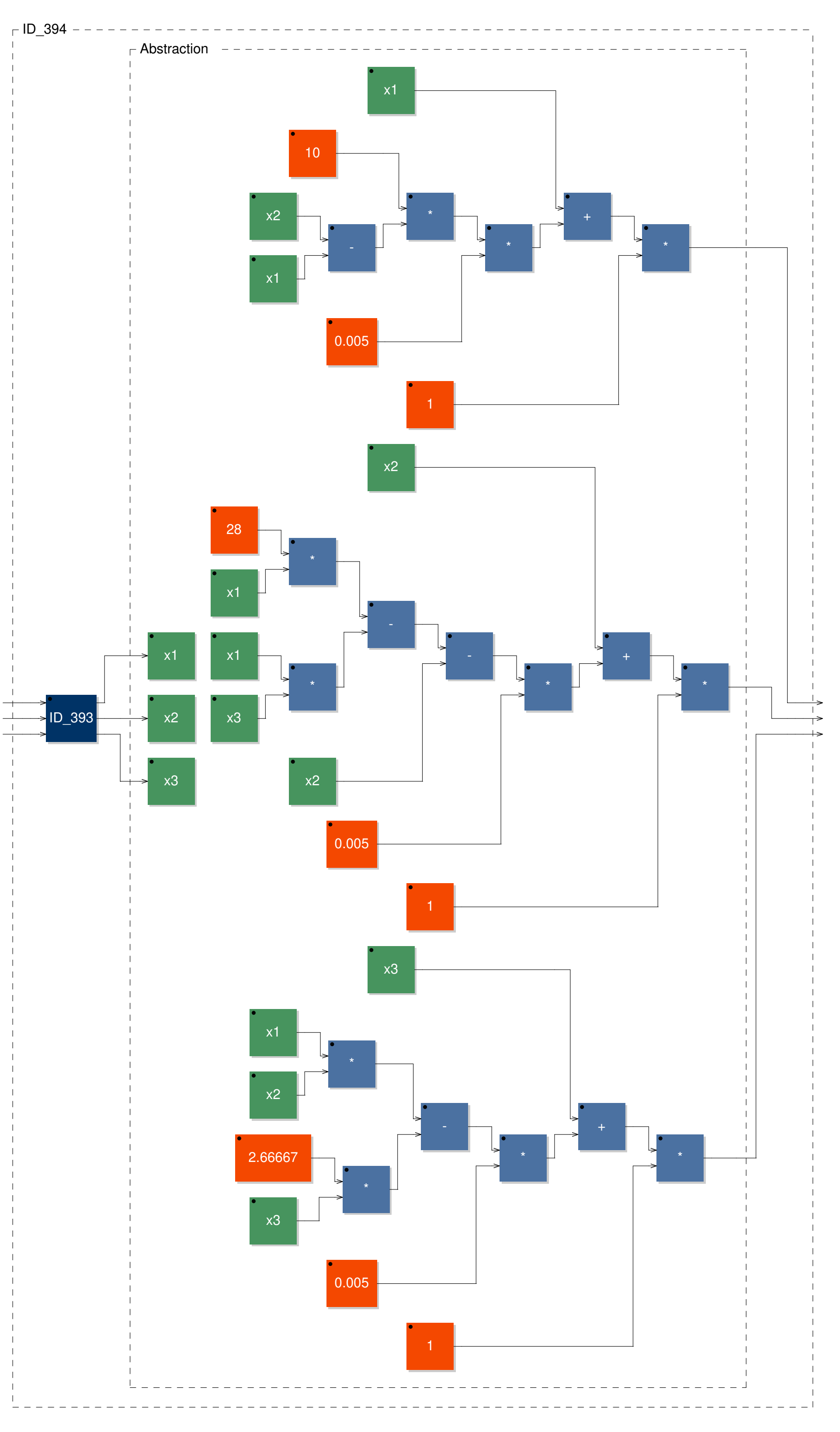


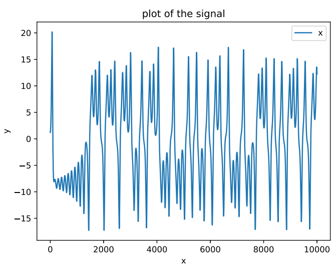
Figure 0.8: Schema Topologico delle equazioni del sistema di Lorenz

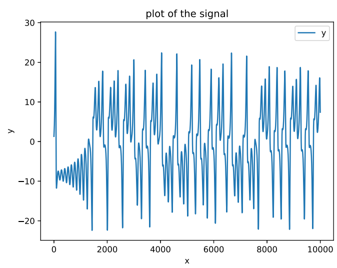
In questa rappresentazione i valori dati sono di x = 1.2, y = 1.3, z = 1.6. Ma nella discretizzazione si rendono necessari altri parametri interni alla topologia esposta, che andiamo ad osservare all'interno del codice in Faust appartenente a questa:

```
--------------------------------------------------------------------------------
Codice Faust: lorenz.dsp
--------------------------------------------------------------------------------
// import Standard Faust library
// https://github.com/grame-cncm/faustlibraries/
import("stdfaust.lib");

// Lorenz System Osc with 3 Out
lorenz(x0,y0,z0) = ( x+_,y+_,z+_ : loop  ) ~ si.bus(3)
   with {

      x = x0-x0';
      y = y0-y0';
      z = z0-z0';

      sigma = 10.0; // a
      rho = 28; // b
      beta = 2.666667; // c
      dt = 0.005; // d

      // iterative times increasing
      q = 1.0;

      loop(x,y,z) =
      (x+(sigma*(y-x))*dt)*q,
      (y+ (rho*x -(x*z) -y)*dt)*q,
      (z+ ((x*y)-(beta*z)) *dt)*q;

   };

routingamp(a) = _*a, _*a, _*a;

process = lorenz(1.2,1.3,1.6);
```

La discretizzazione è stata presa dal paper in Bibliografia:
[Wanqing Song, Jianru Liang - DIFFERENCE EQUATION OF LORENZ SYSTEM. International Journal of Pure and Applied Mathematics Volume 83 No. 1 2013, 101-110], dove vediamo comparire i valori per: dt=0.005; sigma=10.0; rho=28; beta=2.666667. Con in aggiunta il fattore di magnificazione q, che nel codice ho lasciato allo stato q = 1.0 per non alterare il comportamento dell'equazione. Andiamo ora a rappresentare il corrispettivo codice in Python, ed i corrispettivi plots del codice:

```
--------------------------------------------------------------------------------
Codice Python: lorenz.py
--------------------------------------------------------------------------------
# ATTRATTORE DI LORENZ

# definizione dell'equazione differenziale
def lorenz(x, y, z, sigma=10, rho=28, beta=2.666667, dt=0.005, q = 1.0):
   x_0 = (x+(sigma*(y-x))*dt)*q
   y_0 = (y+ (rho*x -(x*z) -y)*dt)*q
   z_0 = (z+ ((x*y)-(beta*z)) *dt)*q
   return x_0, y_0, z_0


# iterazioni del processo (cicli iterativi)
iterazioni = int(1000)

# VALORI DI PARTENZA x,y,z:
x = float(1.2)
y = float(1.3)
z = float(1.6)

# sviluppo e feedback dell'equazione differenziale
for j in range (0,iterazioni):
   # mando i valori nell'equazione differenziale
   out = lorenz(x,y,z)
   # print solo della variabile x
   # per altri outs sostituire con: y,z, o(x,y,z)
   print(x)
   # feedback delle tre variabili x,y,z:
   (x,y,z)=out
```





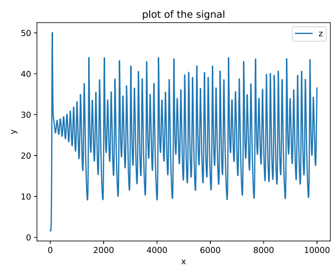


Figure 0.9: Plot grafico per i valori per x di Lorenz in 10.000 iterazioni

Figure 0.10: Plot grafico per i valori per y di Lorenz in 10.000 iterazioni

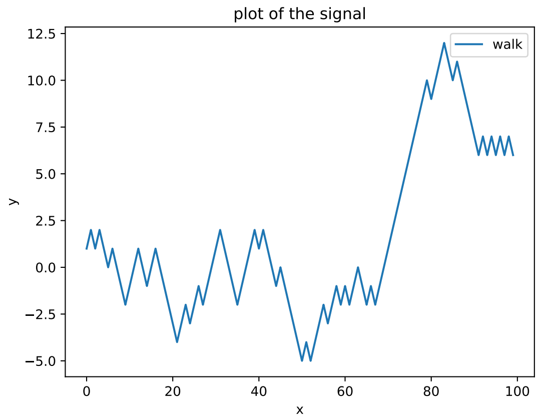
Figure 0.11: Plot grafico per i valori per z di Lorenz in 10.000 iterazioni

VIII. Il Random Walk

In matematica, più precisamente nella teoria della probabilità, un processo stocastico (o processo aleatorio) è la versione probabilistica del concetto di sistema dinamico. Un processo stocastico è un insieme ordinato di funzioni reali di un certo parametro (in genere il tempo) che gode di determinate proprietà statistiche. In generale è possibile identificare questo processo come una famiglia con un parametro di variabili casuali reali X(t) rappresentanti le trasformazioni dallo stato iniziale allo stato dopo un certo tempo t. Uno dei processi stocastici più semplici è la passeggiata aleatoria (random walk), il termine random walk fu introdotto per la prima volta da Karl Pearson nel 1905. Il random walk è la formalizzazione dell'idea di prendere passi successivi in direzioni casuali, ed è il processo markoviano più semplice la cui rappresentazione matematica più nota è costituita dal processo di Norbert Wiener. Un processo di Wiener, che è conosciuto anche come moto Browniano, è un processo stocastico gaussiano in tempo continuo con incrementi indipendenti, ed è usato per modellizzare il moto browniano stesso e diversi fenomeni casuali osservati nell'ambito della matematica applicata, della finanza e della fisica. Con il termine moto browniano si fa riferimento in particolare al moto disordinato di particelle sufficientemente piccole (aventi diametro dell'ordine del micrometro) da essere sottoposte a una forza di gravità trascurabile, presenti in fluidi o sospensioni fluide o gassose (ad esempio il fumo), ed osservabile al microscopio. Il fenomeno fu scoperto agli inizi dell'Ottocento dal botanico scozzese Robert Brown, e modellizzato nel 1905 dal fisico teorico tedesco Albert Einstein. Procediamo nell'implementazione in Python di un random walk ed andiamo ad osservarne il comportamento tramite il plot grafico.

```
--------------------------------------------------------------------------------
Codice Python: randomwalk.py
--------------------------------------------------------------------------------
# RANDOM WALK

from random import random

# PROCESSO:
# la funzione process(a) è il processo deterministico
# su cui si vuole fare - feedback - reiterazione dell'output
# plusminusrand è ad ogni ciclo un -1 o +1 su a
def process(a):
   plusminusrand = ( (random() > 0.5)*2 -1 )
   return a + plusminusrand

# ITERAZIONI:
# è definito qui quante iterazioni ripete il processo stocastico:
iterazioni = int(100)

# VALORE DI PARTENZA:
# valore di partenza del processo stocastico:
valore= float(0)

for j in range (0,iterazioni):
   # mando il valore nel process
   out=process(valore)
   print(out)
   # print( (random() > 0.5)*2 -1 )
   # feedback: - prendo l'uscita da out e la reimmetto nel process
```

```
valore=out
```



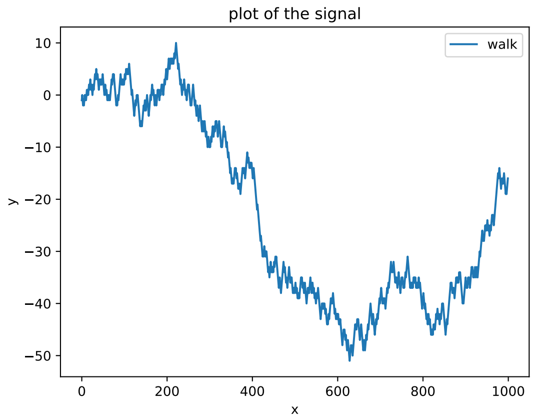


Figure 0.12: Plot grafico del random walk per 100 iterazioni

Figure 0.13: Plot grafico del random walk per 1000 iterazioni

Possiamo notare dai plot di questo semplice processo stocastico, che le iterazioni conducono a valori che potenzialmente possono arrivare sino ad infinito: queste poche iterazioni conducono a valori molto distanti da quelli di partenza, figuriamoci in applicazioni audio dove la frequenza di campionamento è solitamente ad un valore minimo di 44100 campioni al secondo!
Lo stesso problema può ovviamente presentarsi anche con le applicazioni dei sistemi caotici, difatti le equazioni differenziali che abbiamo studiato fino ad ora, non consentono di poter essere esplorate come segnali audio DSP in tutte le regioni della loro esistenza, e per tutti i valori iniziali.
L'esplorazione di queste equazioni in regioni instabili tuttavia è possibile grazie al concetto di Costrizione Matematica. In matematica, una costrizione è una condizione di un problema di ottimizzazione che la soluzione deve soddisfare, in questo modo si possono utilizzare delle soluzioni a delle equazioni differenziali che altrimenti per i nostri scopi sarebbero inutilizzabili.
Partiamo da una semplice costrizione, che ho introdotto all'interno del codice di un random walk implementato in Faust. La costrizione in questo caso avviene tramite la tecnica di Wavefolding. Il Wavefolding è un tipo distorsione che si manifesta quando l'ampiezza di ingresso supera la soglia. In questa tecnica i picchi vengono tagliati, in modo simile al clipping digitale, ma la differenza è che nel wavefolding, i picchi di ampiezza sopra la soglia vengono invertiti di fase in una serie di pieghe: "folds". In bibliografia c'è un paper che espone in maniera esaustiva la tecnica utilizzata fra i vari argomenti: [Jatin Chowdhury - COMPLEX NONLINEARITIES FOR AUDIO SIGNAL PROCESSING. Center for Computer Research in Music and Acoustics Stanford University Palo Alto, CA]
andiamo dunque a visualizzare il corrispettivo codice in Faust del random walk constrained, e la sua topologia.

```
--------------------------------------------------------------------------------
Codice Faust: Constrained_Random_Walk.dsp
--------------------------------------------------------------------------------
// import Standard Faust library
// https://github.com/grame-cncm/faustlibraries/
import("stdfaust.lib");

// RANDOM WALK - function
   randomwalk(walkfreq,noisefreq,noiseseed) = randomwalkout
   with{

      // NOISE GENERATION - function
      noise(seed) = variablenoiseout
      with{
      rescaleint(a) = a-int(a);
      variablenoiseout = ((+(1457932343)~*(1103515245)) * seed)
      / (2147483647.0) : rescaleint;
      };


      // SAMPLE AND HOLD - function
      sampleandhold(frequency) = sampleandholdout
      with{
         // PHASOR
         decimal(x)= x-int(x); // reset to 0 when int
         phase = frequency/ma.SR : (+ : decimal) ~ _; // phasor with frequency
         // PHASOR to 0 and 1
         saw = phase-0.5; // phasor : -0.5 to +0.5
         ifpos = (saw > 0); // phasor positive =1; phasor negative =0
         // PHASOR 1 to Impulse
       trainpulse = ( ifpos - ( ifpos:mem ) ) > 0; // impulse and delette all under 0
         // SAMPLE AND HOLD
         sampleandholdout(a) = (*(1 - trainpulse) + a * trainpulse) ~ _;
      };


         // RANDOM WALK:
         // SAMPLE AND HOLD THE NOISE: noise ---> sample and hold ---> pos
         sahnoise = noise(noiseseed) : sampleandhold(noisefreq);
         // BINARY NOISE (-1 and +1)
         plusminuscond(a) = (a>0)+(a<0)*-1;
         noisebinary = sahnoise : plusminuscond;
         // PHASOR GENERATION
         randomwalkout = (walkfreq/ma.SR)*noisebinary : + ~ _;

   };


// WAVEFOLDING
wavefolding = _ : constrainedout
with{
   intreset(x)= x-int(x);
   triconditionpos(x) = (x<0.5)*(x) + ((x>0.5)*((x*-1)+1));
   trifunctionpos(x) = (x>0)*(x) : triconditionpos;
```

```
    triconditionneg(x) = (x>-0.5)*(x) + ((x<-0.5)*((x*-1)-1));
    trifunctionneg(x) = (x<0)*(x) : triconditionneg;
    constrainedout = (intreset <: trifunctionpos,trifunctionneg :> + : _*2);
};

process = randomwalk(1,3.2,24681242) : wavefolding <: _,_;
```

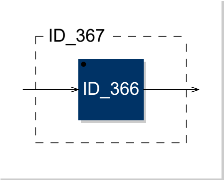


Figure 0.14: Schemi del Wavefolding

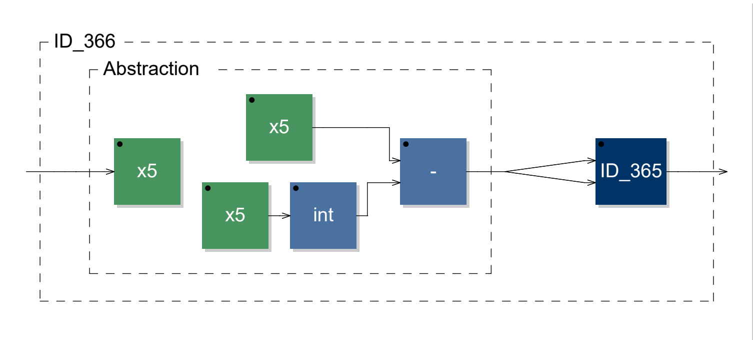


Figure 0.15: Schemi del Wavefolding

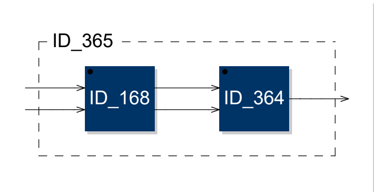


Figure 0.16: Schemi del Wavefolding

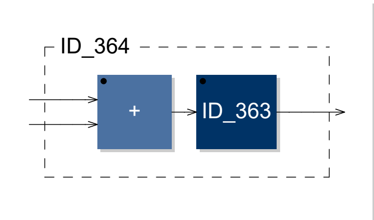


Figure 0.17: Schemi del Wavefolding

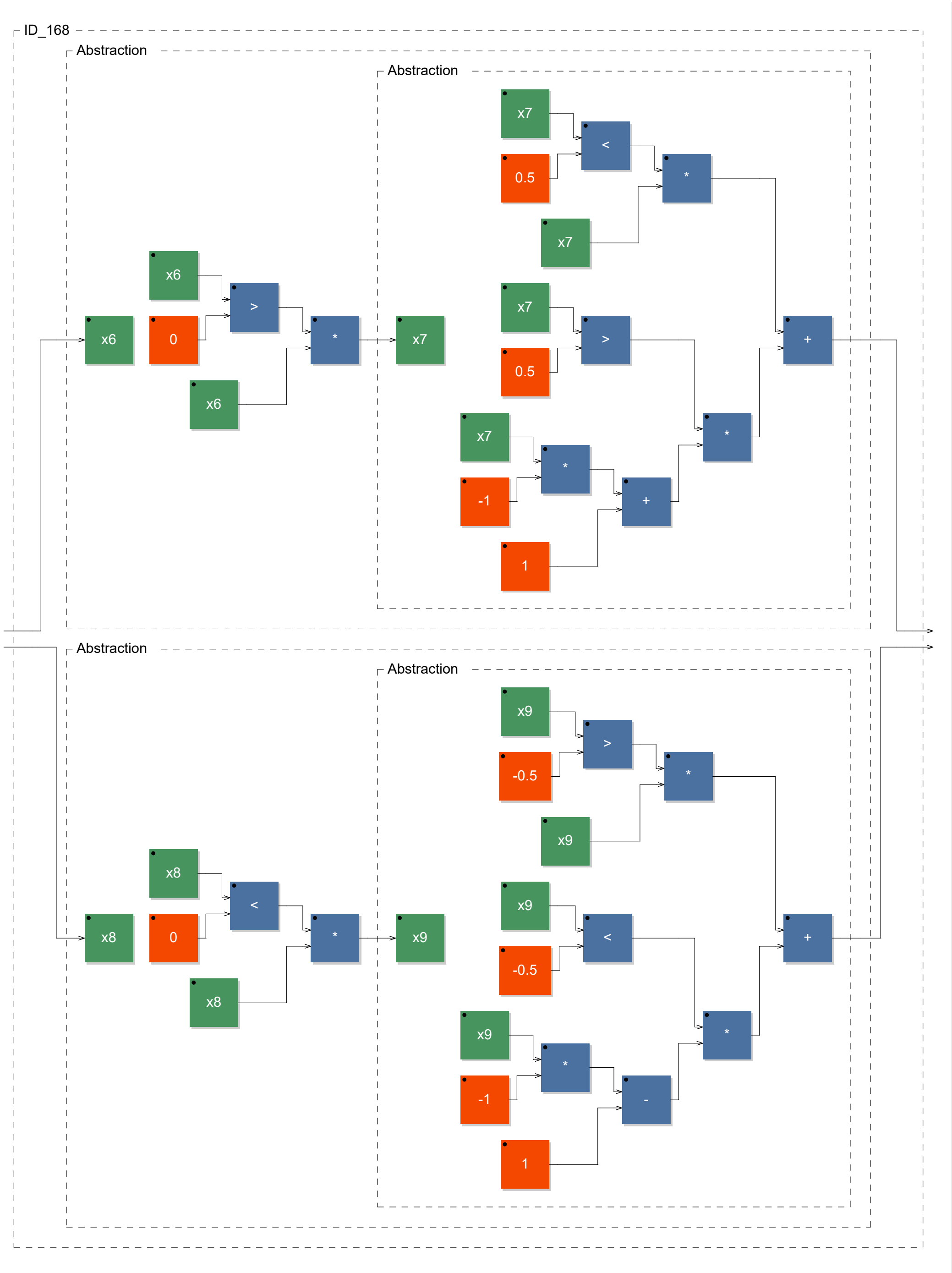


Figure 0.18: Schemi del Wavefolding

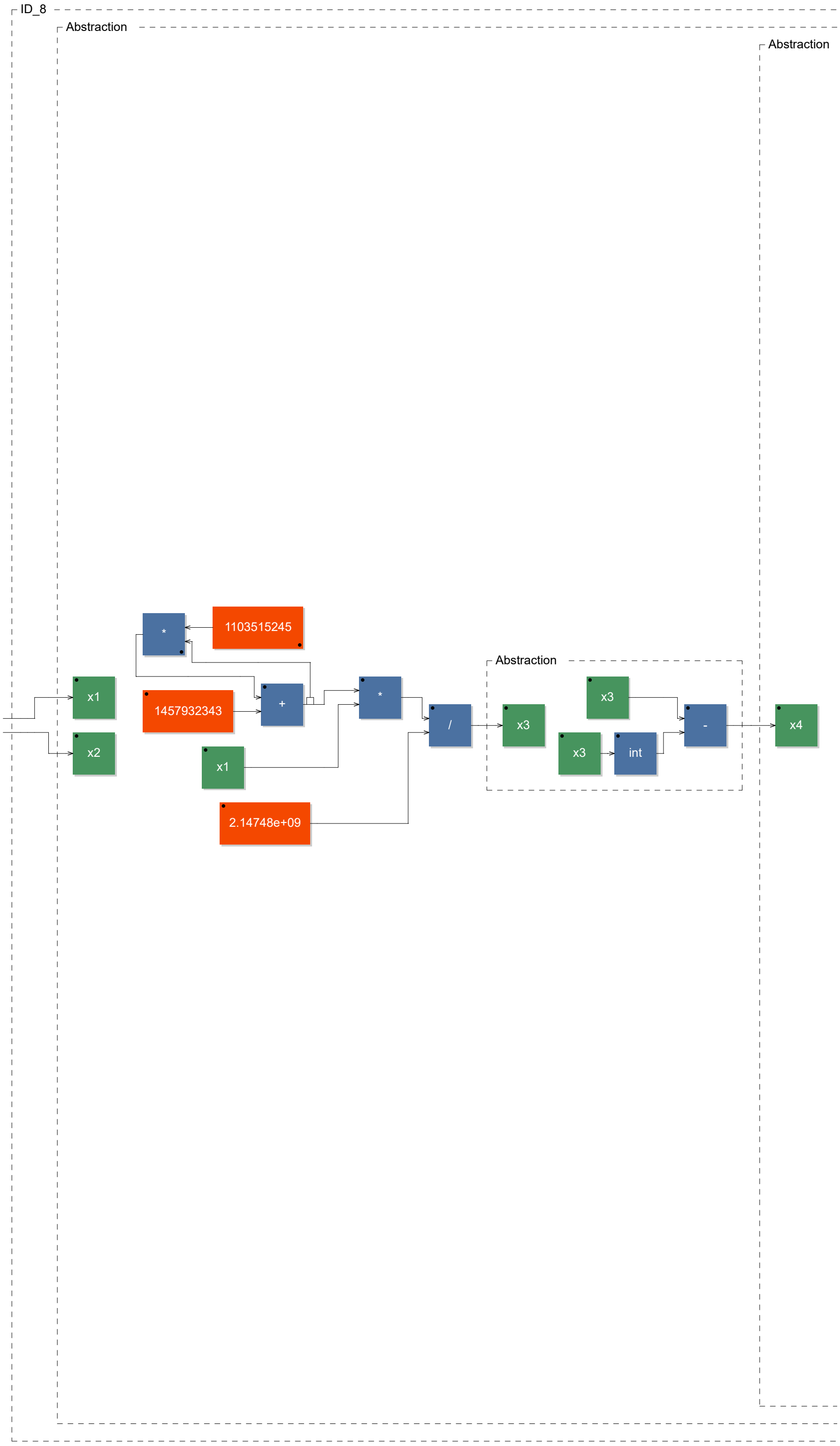


Figure 0.19: Schemi del random walk

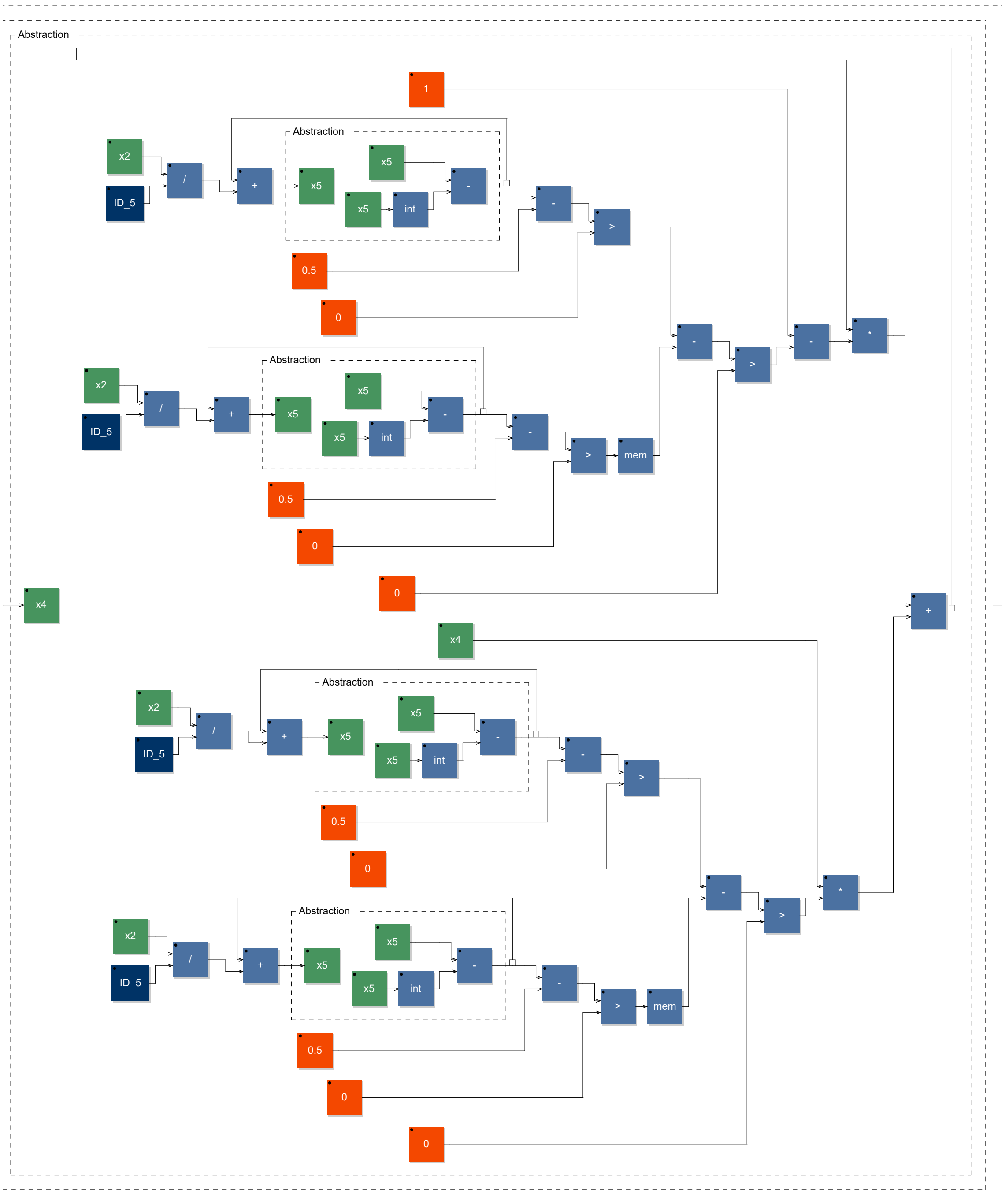


Figure 0.20: Schemi del random walk

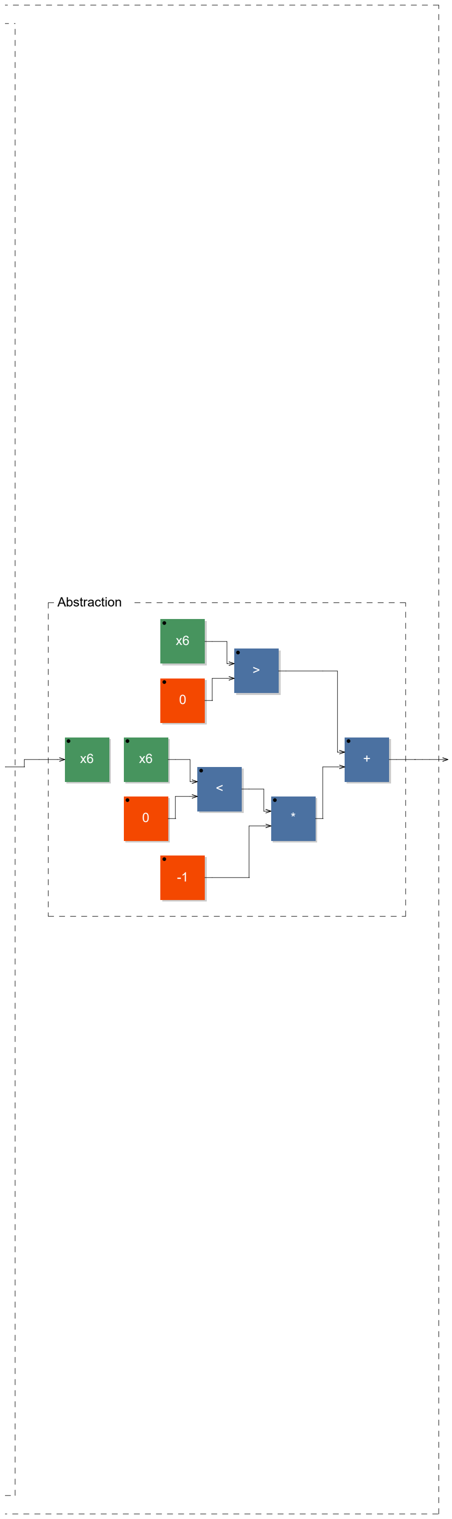


Figure 0.21: Schemi del random walk

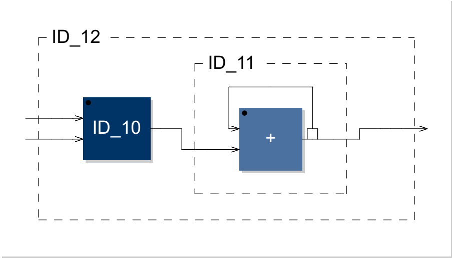


Figure 0.22: Schemi del random walk

IX. Costrizione ed Adattività

Come abbiamo già visto in precedenza, e si può osservare in Bibliografia, esistono recenti ricerche che si pongono il problema di un adattività dei sistemi caotici, col fine di poterli utilizzare per determinati scopi. Per poter esplorare un sistema caotico, col fine di creare un segnale udibile in tutte le sue regioni, è necessario dunque porsi un problema di ottimizzazione. Prima abbiamo introdotto il concetto di costrizione matematica utilizzando il Wavefolding, ora invece andando ad introdurre un problema di ottimizzazione riguardo l'equazione di Lorenz, è necessario chiedersi quali strumenti possano contrastare i problemi che possono verificarsi. Il DC Offset, e i valori più grandi di -1,+1 (fino ad infinito).
Il sistema di Lorenz che sono andato a costruire è una modifica dell'equazione differenziale, che introduce per ogni equazione: un DC Blocker per l'offset, un Softclipper per costringere le regioni fra +/- 1, e un filtro Lowpass One-Pole TPT di Vadim Zavalishin per interventi timbrici.
Il DC blocker è uno strumento indispensabile nella modellazione della guida d'onda digitale e in altre applicazioni. È spesso necessario rimuovere la componente Continua del segnale che circola in una linea di ritardo, consiste sostanzialmente in un filtro FIR highpass, poiché le continue possono essere viste come frequenze a 0Hz, ed un onepole in serie che recupera parte della cancellazione del FIR:

$$y(n) = x(n) - x(n-1) + Ry(n-1), \tag{4}$$

Il Softclipping è invece uno dei più classici saturatori, costituito dalla funzione tangente iperbolica:

$$y(t) = tanhx(t), \tag{5}$$

Anche se il segnale di ingresso di questo saturatore è molto grande, l'uscita non supera mai +/-1. Quindi, questo elemento satura il segnale (origine del termine saturatore).
Infine il TPT One-Pole di Vadim Zavalishin, il motivo per cui questo è così interessante è che preserva matematicamente la topologia del filtro analogico originale, incluso (nella maggior parte dei casi) il ritardo zero campioni. Questa categoria di filtri è chiamata Topology Preserving Transform filters o TPT filters. In Bibliografia un paper di Will Pirkle che ne parla:
[Will Pirkle - Virtual Analog (VA) Filter Implementation and Comparisons, Copyright © 2013 Will Pirkle]
Andiamo ora a vedere il codice del sistema di Lorenz modificato, e a seguito uno schema che illustra le sostanziali modifiche rispetto alla vecchia implementazione del codice Faust.

```
--------------------------------------------------------------------------------
Codice Faust: Lorenz_System_Oscillator.dsp
--------------------------------------------------------------------------------
// import Standard Faust library
// https://github.com/grame-cncm/faustlibraries/
import("stdfaust.lib");

// Lorenz System Synth
lorenzsynth
(X_in,Y_in,Z_in,
Sigma_in,Rho_in,Beta_in,Dt_in,
Qoffset,Qamp,FreqCut) =
(ro.interleave(3,2) : ( x0+_,y0+_,z0+_ : loop  )) ~ si.bus(3)
with{
        loop(x,y,z) =
        ma.tanh(filterTPT(FreqCut,
        ((x+(sigma*(y-x))*dt)*(nonlinearQ) : dcblocker(1,0.98)))),
        ma.tanh(filterTPT(FreqCut,
        (y+ (rho*x -(x*z) -y)*dt)*(nonlinearQ) : dcblocker(1,0.98))),
        ma.tanh(filterTPT(FreqCut,
        (z+ ((x*y)-(beta*z)) *dt)*(nonlinearQ) : dcblocker(1,0.98)));


        // System Variables
        x0 = _;
        y0 = _;
        z0 = _;
        sigma = Sigma_in; // a
        rho = Rho_in; // b
        beta = Beta_in; // c
        dt = Dt_in; // d

        // Noise (positive 0-1)
        noise(seed) = variablenoiseout
        with{
        rescaleint(a) = ((a-int(a))+1)*0.5;
        variablenoiseout = ((+(1457932343)~*(1103515245)) * seed)
        / (2147483647.0) : rescaleint;
        };
        // NON-Linear iterative times increasing
        nonlinearQ = noise(24215682)*Qamp+Qoffset;
        // Filter Section : DC Block & Filter
        dcblocker(zero,pole) = dcblockerout
        with{
        onezero =  _ <: _,mem : _,*(zero) : -;
        onepole = + ~ *(pole);
        dcblockerout = _ : onezero : onepole;
        };
        // TPTfilter - Lowpass Out
        filterTPT(CF, x) = loop ~ _ : ! , si.bus(3) : _,!,!
        with {
        g = tan(CF * ma.PI / ma.SR);
        G = g / (1.0 + g);
        loop(s) = u , lp , hp , ap
```

```
        with {
            v = (x - s) * G;
            u = v + lp;
            lp = v + s;
            hp = x - lp;
            ap = lp - hp;
        };
    };
};


// GUI
Qoffset = hslider("[2] Q-Offset",10,1,100,0.01) : si.smoo;
Qamp = hslider("[3] Nonlinear-Q-Amp",0,0,10,0.01) : si.smoo;
CF = hslider("[4] Lowpass Cut Hz", 1000, 20, 20000, .001): si.smoo;
DiracGUI = button("[0] Dirac");
Dirac = DiracGUI-DiracGUI' > 0;
Analoginput = hslider("[1] Analog Input ",0,0,10,0.01) : si.smoo;
routingamp1(a,b,c) =
a*(hslider("[5] X-OUT1",0.1,0,1,0.01) : si.smoo) +
b*(hslider("[6] Y-OUT1",0,0,1,0.01) : si.smoo) +
c*(hslider("[7] Z-OUT1",0,0,1,0.01) : si.smoo) ;
routingamp2(a,b,c) =
a*(hslider("[8] X-OUT2",0.1,0,1,0.01) : si.smoo) +
b*(hslider("[9] Y-OUT2",0,0,1,0.01) : si.smoo) +
c*(hslider("[9] Z-OUT2",0,0,1,0.01) : si.smoo) ;


process = (_*Analoginput)+(Dirac*0.1)
<: lorenzsynth
(_,_,_, // X,Y,Z
10,28,2.65,0.01, //Sigma, Rho, Beta, Dt
Qoffset,Qamp,CF) <: routingamp1, routingamp2; // GUI
```

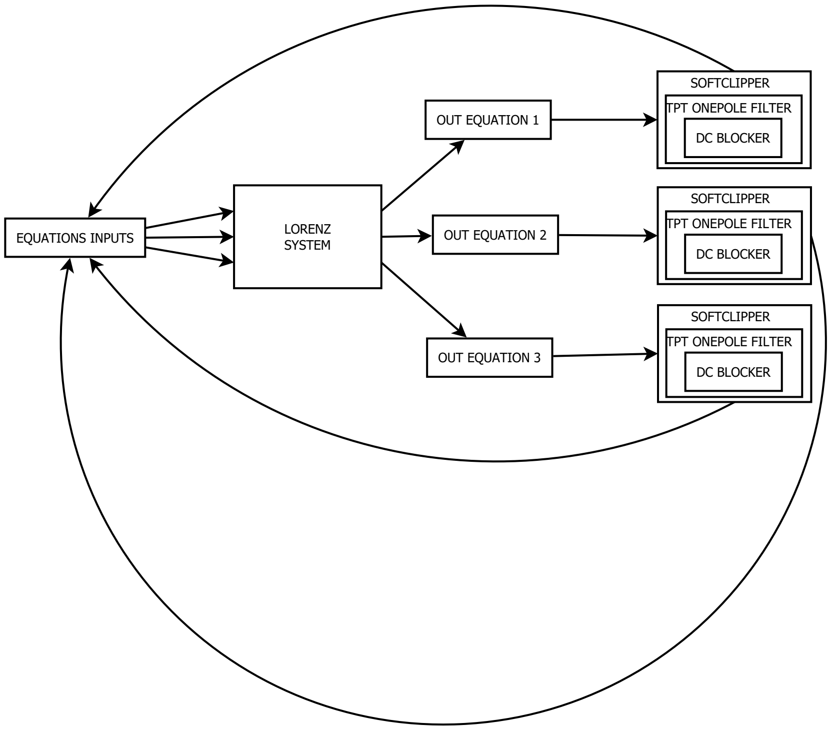


Figure 0.23: Schema di Modifica del sistema di Lorenz
Routing: Equation --> DC Blocker --> TPT Filter --> Softclipper

X. Conclusioni

Come nel feedback acustico, che consiste nel ritorno di un suono fra una sorgente e un recettore, il territorio delle implementazioni digitali dei sistemi caotici discreti, o più in generale dei sistemi complessi, può essere un territorio sonoro d'indagine interessante per la produzione di Musica e Sound Design. La tempo varianza di questi sistemi, la loro imprevedibilità e complessità, può permettere di lavorare in termini di controllo tramite retroazioni negative e positive ottenendo comportamenti più naturali di quelli ottenibili attraverso le tecniche di sintesi tradizionali, ottenendo delle non-linearità grazie alla natura stessa di questi sistemi.

Bibliography

[Chowdhury Jatin] Chowdhury Jatin Center for Computer Research in Music and Acoustics Stanford University Palo Alto, CA COMPLEX NONLINEARITIES FOR AUDIO SIGNAL PROCESSING https://ccrma.stanford.edu/~jatin/papers/Complex_NLs.pdf

[DI SCIPIO AGOSTINO] DI SCIPIO AGOSTINO 'Sound is the interface': from interactive to ecosystemic signal processing, Published online by Cambridge University Press: 21 April 2004 'Sound is the interface': from interactive to ecosystemic signal processing, Published online by Cambridge University Press: 21 April 2004 https://www.ak.tu-berlin.de/fileadmin/a0135/Unterrichtsmaterial/Di_Scipio/Sound_is_the_interface.PDF

[Orpheus Instituut - ECHO: Issue 3 - Feedback] Orpheus Instituut - ECHO: Issue 3 - Feedback, Author(s): Adam Pultz Melbye (ed.) Publication year: 31 January 2022 Orpheus Instituut - ECHO: Issue 3 - Feedback, Author(s): Adam Pultz Melbye (ed.) Publication year: 31 January 2022 https://echo.orpheusinstituut.be/issue/3-feedback

[Pirkle Will] Pirkle Will Virtual Analog (VA) Filter Implementation and Comparisons, Copyright © 2013 Will Pirkle Virtual Analog (VA) Filter Implementation and Comparisons http://www.willpirkle.com/Downloads/AN-4VirtualAnalogFilters.2.0.pdf

[Sanfilippo Dario, June 29th{ July 1st2021] Sanfilippo Dario, June 29th{ July 1st2021 Proceedings of the 18thSound and Music Computing Conference Constrained Differential Equations as Complex Sound Generators https://zenodo.org/record/5040585#.Ygp8fd_MLDc

[Sanfilippo Dario, Valle Andrea] Sanfilippo Dario, Valle Andrea Feedback Systems: An Analytical Framework, Computer Music Journal (2013) 37 (2): 12{27. Feedback Systems: An Analytical Framework, Computer Music Journal (2013) 37 (2): 12{27. https://iris.unito.it/retrieve/handle/2318/142348/23617/Sanfilippo-Valle.pdf

[Song Wanqing] Song Wanqing International Journal of Pure and Applied Mathematics Volume 83 No. 1 2013, 101-110 DIFFERENCE EQUATION OF LORENZ SYSTEM. https://www.ijpam.eu/contents/2013-83-1/9/9.pdf

[Yadegari Shahrokh] Yadegari Shahrokh CHAOTIC SIGNAL SYNTHESIS WITH REAL-TIME CONTROL: SOLVING DIFFERENTIAL EQUATIONS IN PD, MAX/MSP, AND JMAX, Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03), London, UK, September 8-11, 2003 CHAOTIC SIGNAL SYNTHESIS WITH REAL-TIME CONTROL: SOLVING DIFFERENTIAL EQUATIONS IN PD, MAX/MSP, AND JMAX http://yadegari.org/download/YadegariDAFX03.pdf