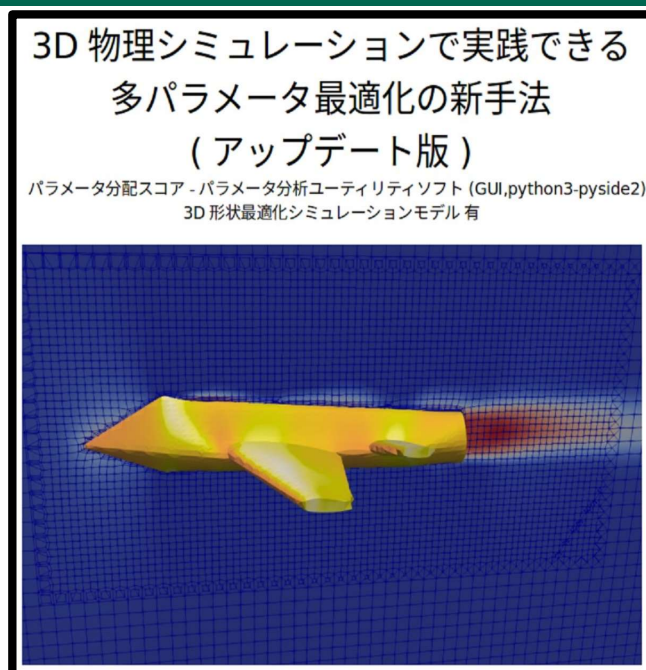


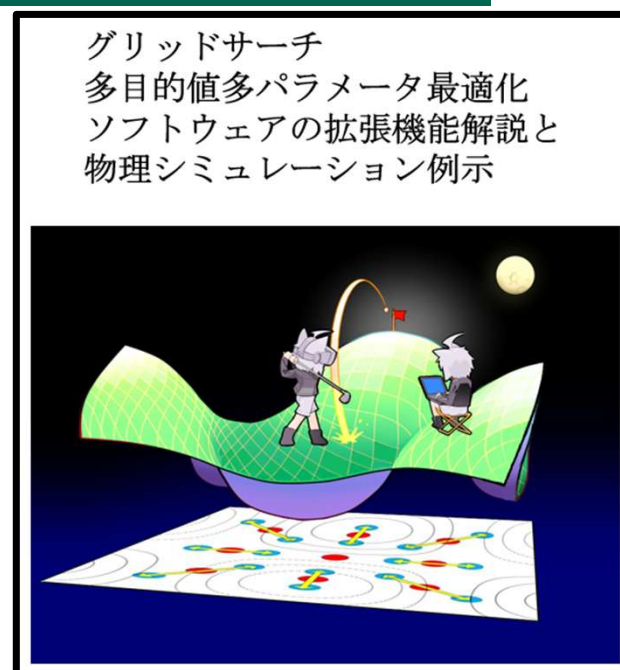
多目的値,多パラメータ最適化ソフト (自作)の回路への適用

17 Nov 2025

AirtomoR
(Kindleも出してます)



3年前位

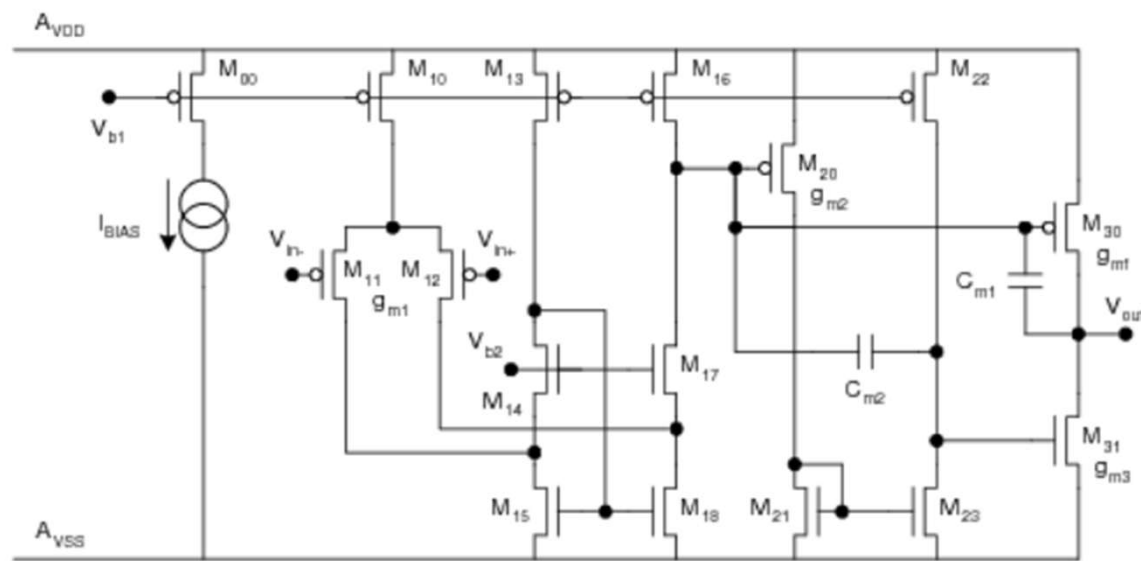


今年5月

目次

- Joao Ramos氏モデル解読
- オペアンプコンテストを参考にした評価系
- python操作内容概要
- グリッドサーチの流れ
- グリッドサーチの水準、組合せパターン種類とtrial数
- (備考)機械学習での寄与度確認
- (備考)CPUコアの並列割振りテクニック

Joao Ramos氏モデル解読



Three stage operational amplifier.

<https://asco.sourceforge.net/examples.html> より

```
# Parameters #
Supply voltage:#VSUPPLY#:3.0:2.4:3.3:LIN_DOUBLE:---
Minimal gate length:#LMIN#:0.35U:0.35U:0.35U:LIN_DOUBLE:---
Bias voltage:#VBIAS#:1.25:1.3:0:LIN_DOUBLE:OPT
Bias current:#IBIAS#:5E-6:1E-6:10E-6:LIN_DOUBLE:OPT
Load capacitance:#CLOAD#:100E-12:100E-12:130E-12:LIN_DOUBLE:---
Load resistance:#RLOAD#:25E3:10E3:50E3:LIN_DOUBLE:---
C compensation 1:#CC1#:15p:2p:20p:LIN_DOUBLE:OPT
C compensation 2:#CC2#:3p:2p:20p:LIN_DOUBLE:OPT
Length group 1:#LM1#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 2:#LM2#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 3:#LM3#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 4:#LM4#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 5:#LM5#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 6:#LM6#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 7:#LM7#:0.5E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Width M00_10:#WM00_10#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M11_12:#WM11_12#:40E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M13+16:#WM13_16#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M14_17:#WM14_17#:6E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M15_18:#WM15_18#:11.01E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M20:#WM20#:15E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M22:#WM22#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M21_23:#WM21_23#:2E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M30:#WM30#:1.5E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M31:#WM31#:1.5E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
```

cfgファイル抜粋(パラメータ上下限)

Ramos氏は自作のパラメータ最適化ツールの仕様として.cfgファイルという中でオペアンプのFETゲート幅、長、コンデンサ容量等で振る値の上下を規定(次ページのテキストベース回路図の未定数にtrial毎に代入する前提と思われる)

Joao Ramos氏モデル解読

```
1  *Three stage operational amplifier
2
3  *** ** OPAMP SUBCIRCUIT *** **
4  .SUBCKT PFC.SUB VP VN VOUT IBIAS VB1 AVDD AVSS
5
6  M00 IBIAS IBIAS AVDD AVDD PMOS W=#WM00_10# L=#LM1#
7
8  * differential pair
9  M10 1 IBIAS AVDD AVDD PMOS W=#WM00_10# L=#LM1# M=6
10
11 M11 2 VN 1 1 PMOS W=#WM11_12# L=#LM2#
12 M12 3 VP 1 1 PMOS W=#WM11_12# L=#LM2#
13
14 * folded cascode
15 M13 4 IBIAS AVDD AVDD PMOS W=#WM13_16# L=#LM1# M=3
16 M16 5 IBIAS AVDD AVDD PMOS W=#WM13_16# L=#LM1# M=3
17
18 M14 4 VB1 2 AVSS NMOS W=#WM14_17# L=#LM3#
19 M17 5 VB1 3 AVSS NMOS W=#WM14_17# L=#LM3#
20
21 M15 2 4 AVSS AVSS NMOS W=#WM15_18# L=#LM4#
22 M18 3 4 AVSS AVSS NMOS W=#WM15_18# L=#LM4#
23
24 * second stage
25 M20 6 5 AVDD AVDD PMOS W=#WM20# L=#LM6#
26 M22 7 IBIAS AVDD AVDD PMOS W=#WM22# L=#LM1#
27
28 M21 6 6 AVSS AVSS NMOS W=#WM21_23# L=#LM5#
29 M23 7 6 AVSS AVSS NMOS W=#WM21_23# L=#LM5#
```

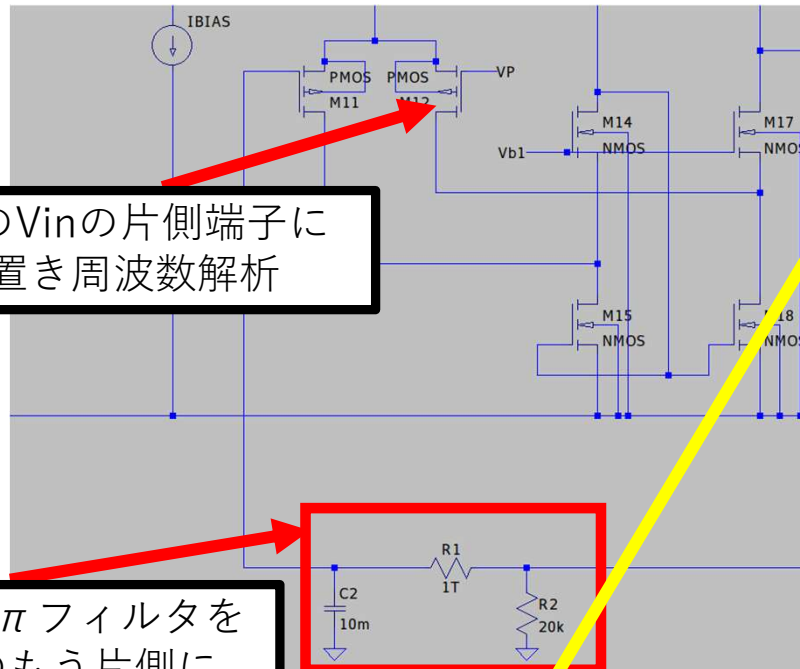
```
31  * third stage
32 M30 VOUT 5 AVDD AVDD PMOS W=#WM30# L=#LM6# M=22
33 M31 VOUT 7 AVSS AVSS NMOS W=#WM31# L=#LM7# M=5
34
35 * compensation
36 CM1 5 VOUT #CC1#
37 CM2 5 7 #CC2#
38 .ENDS PFC.SUB
39
40 *** ** SUPPLY VOLTAGES *** **
41 VDD VDD 0 #VSUPPLY#
42 VSS VSS 0 0
43
44 *** ** BIAS VOLTAGE *** **
45 VVB1 VB1 VSS DC #VBIAS#
46
47 *** ** BIAS CURRENT *** **
48 IIBIAS IBIAS VSS #IBIAS#
49
50 *** ** SUB-CIRCUIT *** **
51 XOPAMP VP VN VOUT IBIAS VB1 VDD VSS PFC.SUB
52
53 *** ** LOAD *** **
54 RL VOUT VX #RLOAD#
55 CL VOUT VX #CLOAD#
56 VX VX VSS '#VSUPPLY#/2'
57
58 *** ** AC LOOP *** **
59 VIN VP VSS '#VSUPPLY#/2' AC 1
```

```
60 RX VN VOUT 1m AC=1E12
61 CX VN VSS 10
62
63 *** ** ANALYSIS *** **
64 .AC DEC 100 0.001 1E9
65 *.PZ V(VOUT) VIN
66 *.PROBE AC VDB(VOUT)
67 *.PROBE AC VP(VOUT)
68 *.OP
69 .INCLUDE p.typ
70 .INCLUDE n.typ
71 .control
72 run
73 .endc
```

メールでも少しやりとりさせて頂いたが
どうやらGUIとして回路図を作って(残して)はない模様
ngspiceだと.spファイル、LTSpiceだと.netファイルで上記テキストのみ
(自分はLTSpiceで評価系等調整したいので論文を参照して構築)

オペアンプコンテストを参考にした評価系

参考： <https://www.opamp-contest.org/2024/sim.html>



オペアンプのVinの片側端子にAC電圧源を置き周波数解析

Voutに対し π フィルタを通してVinのもう片側にフィードバック帰還

利得は0.1Hzの際の利得(V_{out}/V_p)で評価

消費電力は電流値の周波数に対するRMS(積分値相当)で評価



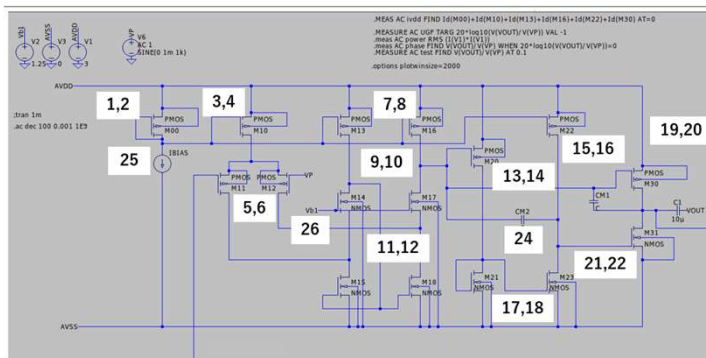
位相余裕は利得0dBの際のVout位相遅れで評価

python 操作内容概要

.asc

.net

.log(.raw経由からも可)



変数編集コード

```
with open('./PREamp3.net',encoding='CP932') as f:
    s = f.read()
    lines = s.split("\n")
    lines[8] = lines[8] + " + " + "Wn" + str(param_list[0]) + " + " + "Ln" + str(param_list[1])
    lines[9] = lines[9] + " + " + "Wn" + str(param_list[2]) + " + " + "Ln" + str(param_list[3]) + " + " + "Mn0"
    lines[10] = lines[10] + " + " + "Wn" + str(param_list[4]) + " + " + "Ln" + str(param_list[5])
    lines[11] = lines[11] + " + " + "Wn" + str(param_list[4]) + " + " + "Ln" + str(param_list[5])
    lines[12] = lines[12] + " + " + "Wn" + str(param_list[6]) + " + " + "Ln" + str(param_list[7]) + " + " + "Mn3"
    lines[13] = lines[13] + " + " + "Wn" + str(param_list[6]) + " + " + "Ln" + str(param_list[7]) + " + " + "Mn3"
    lines[14] = lines[14] + " + " + "Wn" + str(param_list[8]) + " + " + "Ln" + str(param_list[9])
    lines[15] = lines[15] + " + " + "Wn" + str(param_list[8]) + " + " + "Ln" + str(param_list[9])
    lines[16] = lines[16] + " + " + "Wn" + str(param_list[10]) + " + " + "Ln" + str(param_list[11])
    lines[17] = lines[17] + " + " + "Wn" + str(param_list[10]) + " + " + "Ln" + str(param_list[11])
    lines[18] = lines[18] + " + " + "Wn" + str(param_list[12]) + " + " + "Ln" + str(param_list[13])
    lines[19] = lines[19] + " + " + "Wn" + str(param_list[12]) + " + " + "Ln" + str(param_list[13])
```

```
IBIAS N001 AVSS
CM2 N007 N002 C
CM1 N002 N006 C
V1 AVDD 0 3
V2 Vb1 0 1.25
V3 AVSS 0 0
V6 VP 0 SINE(0 1m 1k) AC 1
M00 N001 N001 AVDD AVDD PMOS
M10 N003 N001 AVDD AVDD PMOS
M11 N009 N004 N003 N003 PMOS
M12 N008 VP N003 N003 PMOS
M13 N005 N001 AVDD AVDD PMOS
M16 N002 N001 AVDD AVDD PMOS
M14 N005 Vb1 N009 AVSS NMOS
M15 N009 N005 AVSS AVSS NMOS
M17 N002 Vb1 N008 AVSS NMOS
M18 N008 N005 AVSS AVSS NMOS
```

```
with open('./temp/amp3_1.log',encoding='CP932') as f:
    s = f.read()
    lines = s.split("\n")
    test_result_list1 = []
    print(lines[7],lines[8])
    for i in range(len(lines)):
        if lines[i][6] == "power:":
            print("power found")
            test_result_list1= re.findall(pattern, lines[i] )
            print(test_result_list1)

    test_result1 = test_result_list1[1]
    #test_result1=1000.0*(10**(0.05*float(test_result1)))
    test_result1=1000*float(test_result1)
    test_result1=str(test_result1)
```

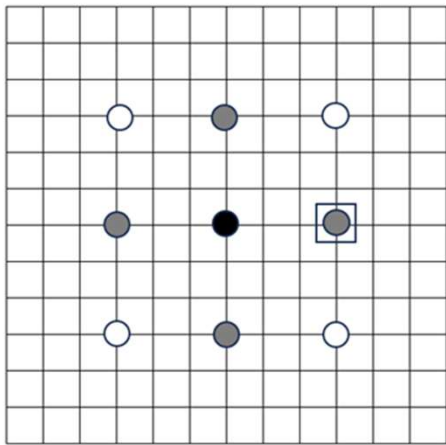
スコア取得コード

本来なら一番上流の.ascファイルで値を与えられるのが自然。
ただモデル作成時にはFETのサブ属性変数にspicelib(spice系libraryで一番メジャー)からは
アクセスできなかったため(作者からまだ未実装との回答、netファイルに出力された値を強引に上書きしている

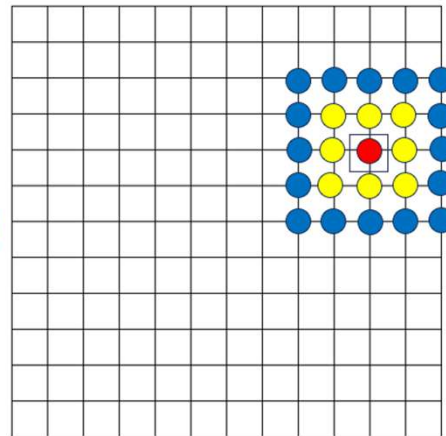
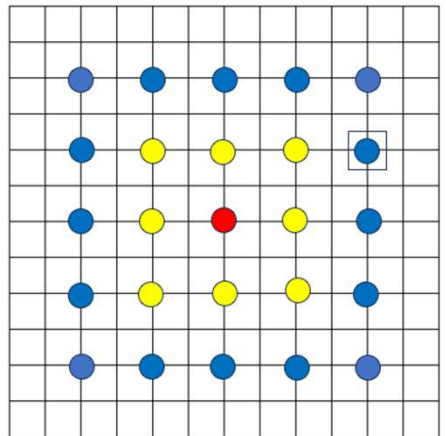
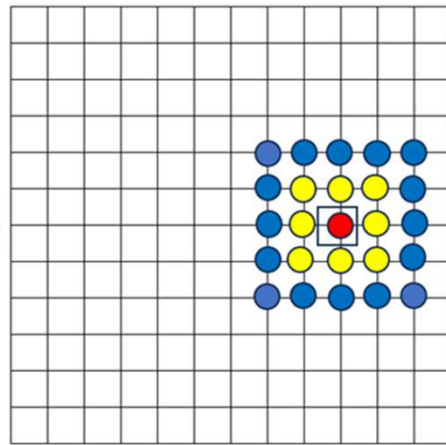
ゲインは単純にVout/Vin、消費電力は電圧値一定と考え電流値そのもの
位相余裕はオペアンプのIn電圧の片方をspiceのAC解析電圧源にして
Voutの利得が0dBになった点とする

グリッドサーチの流れ(数百trial内の想定)

Coarse grid



Fine grid

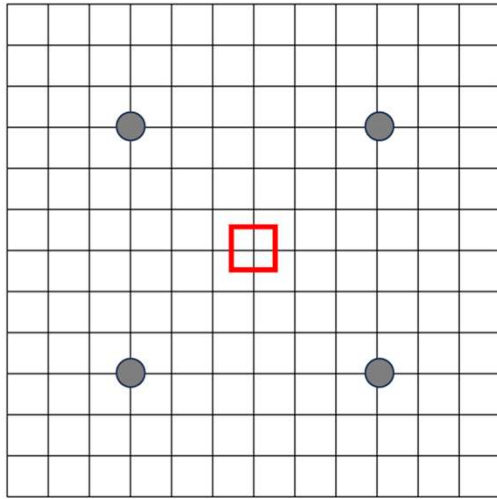


4分割格子walking(2,3水準)、12分割格子walking(3,5水準)
6分割格子walking(3,5水準)、12分割格子walking(3,5水準)
のどちらかが大概良い結果となる
Kindleでは下側を実施したので今回は上側を試してみる

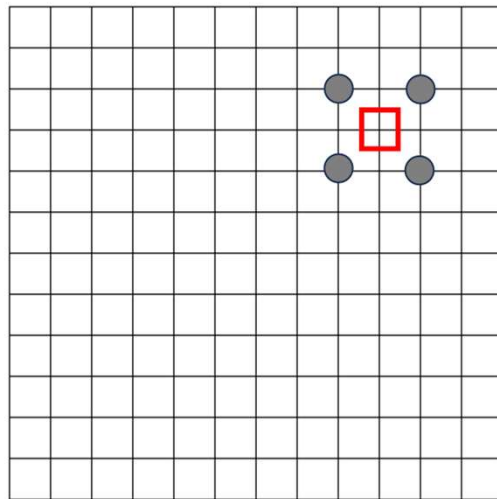
グリッドサーチの水準、組合せパターン種類とtrial数

[1].2水準

Coarse grid



Fine grid



□ 起点 ● 遷移候補先

Pattern,パラメータ長とtrial数の関係

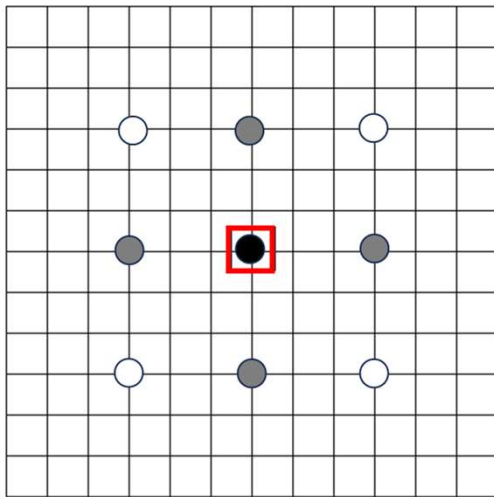
Pattern	8param	32param	128param
logcode	8	12	16
4C2	6	6	6

試行数が少ないのでパラメータ境界条件が妥当かどうかの確認に使用することを想定
(パラメータ数が増えたとそもそも境界条件がイマイチ、という確率が上がるので)

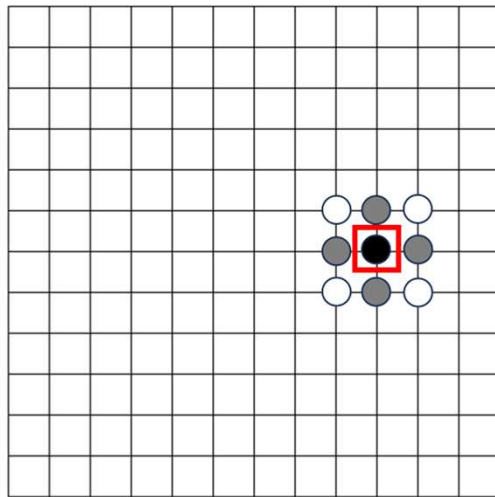
グリッドサーチの水準、組合せパターン種類とtrial数

[2].3水準

Coarse grid



Fine grid



□ 起点 遷移確率

●	○
1/4	1/16

Pattern,パラメータ長とtrial数の関係

Pattern	8param	32param	128param
logcode	32	72	128
4C2	18	18	18

3水準だとLogcodeでも4C2等でも正規分布的

グリッドサーチの水準、組合せパターン種類とtrial数

[3-1].2水準の任意2項目網羅パターンの生成方法

		decimal-represent(assign 0-15 to num-of-param)															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
binary-represent (normal)	1st-digit	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	2nd-digit	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	3rd-digit	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	4th-digit	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	5th-digit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
binary-represent (inverse)	1st-digit	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
	2nd-digit	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	3rd-digit	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	4th-digit	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	5th-digit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

use each row as pattern(10items)

私がlogcodeと名付けたもの
(param長nに対し試行数log n)
実質waveletのHaar基底であるが
これで任意2項目に対し
低水準(≡0),高水準(≡1)の全組合せが
どれかのPatternに発現する

[3-2].2水準の任意3項目網羅パターンの生成方法

(上表の行から任意選択) 0101010101010101 ..

+

(上表の行から任意選択) 0011001100110011 ..

||

単純加算結果を集積 0112011201120112 ..

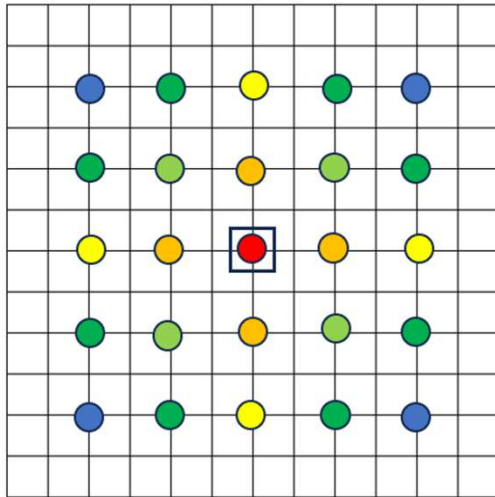
3水準は2水準の全組合せ全部の組合せを足し
合わせて生成(上記例なら10 x 10 / 2(重複分))
0が低水準、1が中水準、2が高水準と見做す

任意2項目網羅はパラメータ数nに対し2水準でlog n, 3水準だと(log n)² / 2, 5水準で(log n)⁴ / 4

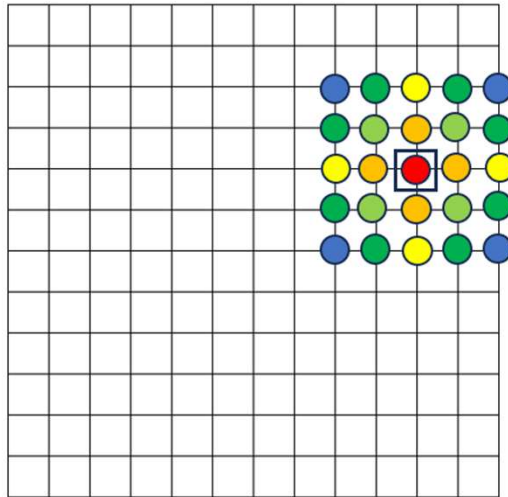
グリッドサーチの水準、組合せパターン種類とtrial数

[4].5水準

Coarse grid



Fine grid



Pattern,パラメータ長とtrial数の関係

Pattern	8param	32param	128param
logcode	224	984	4644
4C2	84	84	84

□ 起点

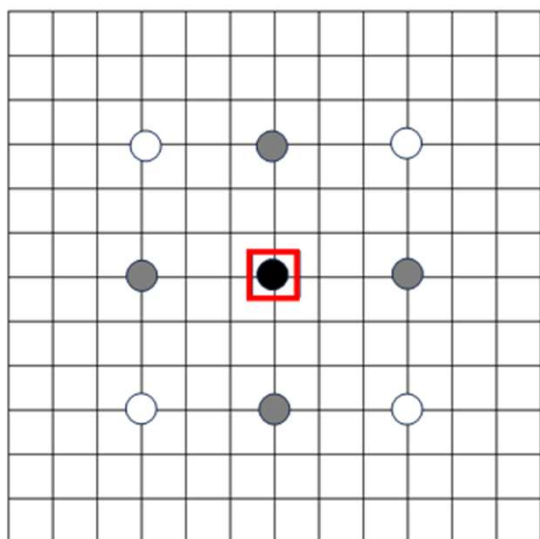
遷移確率
(大凡)

● 9/64 ● 3/32 ● 3/128 ● 1/16 ● 1/64 ● 1/256

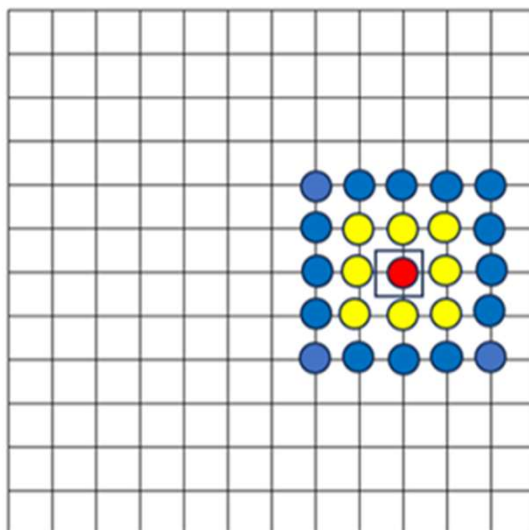
5水準は3水準の全組合せ全部の組合せを足し合わせて生成
5水準でも正規分布的 (4C2はやや正規分布より遠巻きの確率が高い)

(備考)機械学習の補足事項

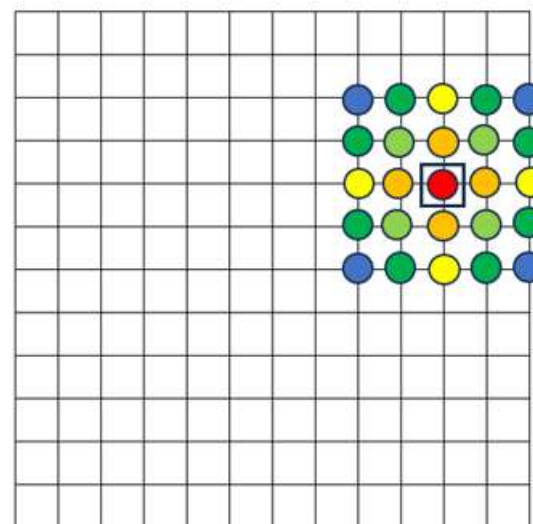
中点->div4ノード群



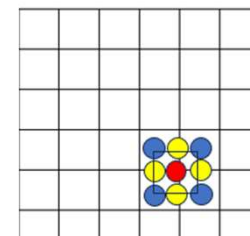
div4終点->div12ノード群



div6終点->div12ノード群



div24ノード群



機械学習は外れ値があると精度がさがるので有望ノード下の近傍子ノードだけで解析できる様にしてる
中点 -> 4分割格子、4分割格子 -> 12分割、12分割格子 -> 24分割で生成したノード等を解析できる
必要データ量はパラメータ長 n に対し $n * \log n$ であるのと線形独立性が必要で、その点には注意
(あと、用意している網羅パターンはそのままだと周期立ってて線形独立に不適の場合あり
各パラメータのup,downを乱数により1/2の確率で反転できるmodulationという機能も用意している)

(備考)CPUコアの並列割振りテクニック

- 内側並列の設定方法

/home/【user名】の直下にmpiHostとmpiRankというファイルを下記要領の内容 (外側2並列,内側3並列の例)のテキストを記載して保存する。(外側の3についてはツリーGUIまたはテーブルGUIのnCase欄にも記載。仮にnCase欄を2,1などに減らせば、実行される外側並列の数はそれに合わせて減る。

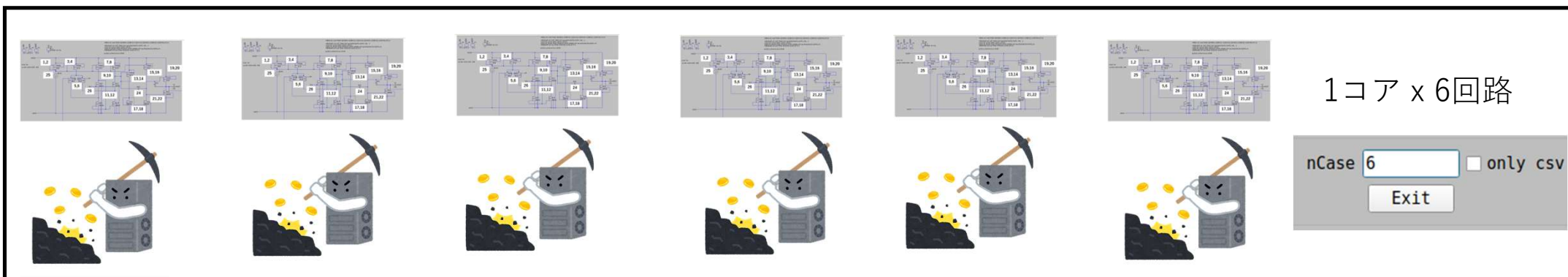
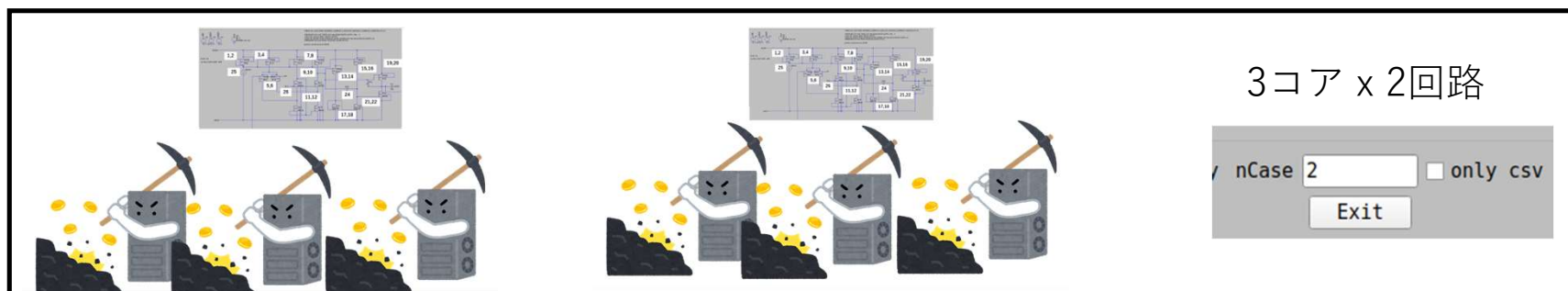
~/mpiHostへの記載例

localhost slots=6

~/mpiRankへの記載例

rank 0=localhost slot=0-2

rank 1=localhost slot=3-5



来週の**11/24**の**Ubuntu + LTSpice + my**ソフト環境
セット**try**を希望する方募集
(事前に**Ubuntu22.04 or 24.04**はセット願)

再来週以降の**ngspice**例題でモデル持ってて
演習に使わせて貰える方募集