x^0.45

2018年4月22日 17:43

第一部分 从式子到程序的输入

两种方式均可，可对比取用效果更好的，其中次数(n)和精度(m)可根据需要选择

1. Bernstein

设 n = 4, 使用钱老师的 MATLAB 程序得到

Bernstein 系数: 0.1298, 0.7671, 0.6128, 0.9502, 0.9883

$$x^{0.45} \approx 0.1298\binom{4}{0}x^4 + 0.7671\binom{4}{1}x^3(1-x) + 0.6128\binom{4}{2}x^2(1-x)^2 + 0.9502\binom{4}{3}x(1-x)^3 + 0.9883\binom{4}{4}(1-x)^4$$

取精度 m = 8, $2^8 = 256$

$$0.1298\binom{4}{0} \approx \frac{33}{256}$$
$$0.7671\binom{4}{1} \approx \frac{786}{256}$$
$$0.6128\binom{4}{2} \approx \frac{941}{256}$$
$$0.9502\binom{4}{3} \approx \frac{973}{256}$$
$$0.9883\binom{4}{4} \approx \frac{253}{256}$$

Vector: [33 786 941 973 253]

文本文件 vector.txt 内容如下：

1 8
4
33 786 941 973 253

2. Maclaurin

在 x = 0 处无展开式; 使用 x = 1 (使用 Wolframalpha 得到)

n = 4

$$x^{0.45} \approx 1 + 0.45(x-1) - 0.12375(x-1)^2 + 0.0639375(x-1)^3 - 0.0407602(x-1)^4$$

设 $t = 1 - x$

$$(1-t)^{0.45} \approx 1 - 0.45t - 0.12375t^2 - 0.0639375t^3 - 0.0407602t^4$$

转换为 BCP, 由于 BCP 与 Bernstein 多项式等价，可使用钱老师的 MATLAB 程序将上式转换为 Bernstein 多项式:

$$(1-t)^{0.45} \approx 0.9990\binom{4}{0}t^4 + 0.8900\binom{4}{1}t^3(1-t) + 0.7502\binom{4}{2}t^2(1-t)^2 + 0.5920\binom{4}{3}t(1-t)^3 + 0.2926\binom{4}{4}(1-t)^4$$

将 x = 1 - t 代回,

$$x^{0.45} = 0.2926\binom{4}{4}x^4 + 0.5920\binom{4}{3}x^3(1-x) + 0.7502\binom{4}{2}x^2(1-x)^2 + 0.8900\binom{4}{1}x(1-x)^3 + 0.9990\binom{4}{0}(1-x)^4$$

取精度 m = 8

$$0.2926\binom{4}{4} \approx \frac{75}{256}$$
$$0.5920\binom{4}{3} \approx \frac{606}{256}$$
$$0.7502\binom{4}{2} \approx \frac{1152}{256}$$
$$0.8900\binom{4}{1} \approx \frac{911}{256}$$
$$0.9990\binom{4}{0} \approx \frac{256}{256}$$

Vector: [75 606 1152 911 256]

vector.txt:
1 8
4
75 606 1152 911 256

第二部分 使用程序

1. 运行主程序

运行的时候请确保目录结构为

```
./                          # 工作目录
|-- CA-w-h.exe              # 主程序
|-- mvsis50720.exe          # 主程序会调用它，程序名是 hard code
|-- master.mvsisrc          # mvsis 程序会读取 其实是空文件
|                           # 没有的话会多输出一行字影响主程序读取
|-- pla/                    # 存放结果的目录 目录名是 hard code
|   |-- case-00/            # 分别存放不同 case 的结果 数字范围是 00 到 99
|   |-- case-01/            # vector.txt中有多少个case就建多少个文件夹
|   `-- ......              # 可以使用脚本批量生成
`-- vector.txt              # 程序的输入 文件名是 hard code
```

一般来说 vector.txt 只有一个 case, 因此可以只建立 "case-00" 这个文件夹.

然后在命令行运行 "CA-w-h.exe". 其中 w 和 h 是编译时选用的不同参数，详情可以参阅论文。

运行后会在 "case-00" 文件夹下面得到至少一个解，命名为 "solution-000.pla", 格式类似于:

.i 12
.o 1
000000000000 1
000100000000 1
……
111111111111 1
.e

".i 12" 表示输入有12个, ".o1" 表示输出有1个；这12个输入中，前面4个为x输入，即我们真正的输入变量——四个独立的，其概率为x的随机变量；后面8个是y输入，是为了控制精度引入的8个独立的，概率为1/2的随机变量。

2. 使用 abc 处理

将 abc 程序，abc.rc 文档, mcnc.genlib 文档, 以及 .pla 文档放在同一个文件夹下；在命令行运行 abc 程序，进入 abc 的交互式界面。

然后用命令 "read_genlib mcnc.genlib" 读取 genlib；用命令 "read xxxxxx.pla" 读取逻辑电路。接下来使用命令 "clp; sop; fx; st; dch; b; map;" 处理电路。最后使用命令 "write_eqn xxxxxx.eqn" 输出逻辑电路的等式格式。

eqn 格式形如：

```
# Equations for "gamma_correction_maclaurin" written by ABC on Sun Apr 22 18:20:55 2018
INORDER = x00 x01 x02 x03 x04 x05 x06 x07 x08 x09 x10 x11;
OUTORDER = z0;
n14 = !x09 * !x08;
n15 = !x10 * !x05;
n16 = !x07 + (!x11 * !x06);
n17 = !x01 + !n14 + !n16 + !n15;
n18 = x04 * x00;
n19 = !n18 + !n17;
n20 = !n19 + !x02;
n21 = !x08;
n22 = !x07 * !x06;
n23 = !n22 + !n21;
n24 = !n23 + !x05;
n25 = !x04 * !x00;
n26 = !x03 * (!n25 + !n24);
n27 = !n14 + !n16 + !n15;
n28 = !n27;
n29 = !n22 * !x00;
n30 = !x07 + !x01;
n31 = !n28 + !n30 + !n29;
n32 = !x02;
n33 = !x00 + (!x08 * !n32);
n34 = !x05 + (!x09 * !x08);
n35 = !x01 + !n22 + !n34 + !n33;
z0 = !n20 + !n26 + !n35 + !n31;
```

其中 x00 到 x03 是 x 输入， x04 到 x11 是 y 输入，n开头的是电路的中间节点，z0 是输出。接下来可以写代码来验证或者使用这个电路。下面提供一个我的 python 脚本，与这个 eqn 对应。

```
import random
import sys

x = float(sys.argv[1])
y = 0.5

def gen_stochastic_bit(r):
    if random.uniform(0, 1) < r:
        return True
    else:
        return False

def eval(x, y):
    x00 = gen_stochastic_bit(x)
    x01 = gen_stochastic_bit(x)
    x02 = gen_stochastic_bit(x)
    x03 = gen_stochastic_bit(x)
```

```
    x04 = gen_stochastic_bit(y)
    x05 = gen_stochastic_bit(y)
    x06 = gen_stochastic_bit(y)
    x07 = gen_stochastic_bit(y)
    x08 = gen_stochastic_bit(y)
    x09 = gen_stochastic_bit(y)
    x10 = gen_stochastic_bit(y)
    x11 = gen_stochastic_bit(y)
    n14 = (not x09) and (not x08)
    n15 = (not x10) and (not x05)
    n16 = (not x07) or ((not x11) and (not x06))
    n17 = (not x01) or (not n14) or (not n16) or (not n15)
    n18 = x04 and x00
    n19 = (not n18) or (not n17)
    n20 = (not n19) or (not x02)
    n21 = (not x08)
    n22 = (not x07) and (not x06)
    n23 = (not n22) or (not n21)
    n24 = (not n23) or (not x05)
    n25 = (not x04) and (not x00)
    n26 = (not x03) and ((not n25) or (not n24))
    n27 = (not n14) or (not n16) or (not n15)
    n28 = (not n27)
    n29 = (not n22) and (not x00)
    n30 = (not x07) or (not x01)
    n31 = (not n28) or (not n30) or (not n29)
    n32 = (not x02)
    n33 = (not x00) or ((not x08) and (not n32))
    n34 = (not x05) or ((not x09) and (not x08))
    n35 = (not x01) or (not n22) or (not n34) or (not n33)
    z0 = (not n20) or (not n26) or (not n35) or (not n31)
    return z0

sum = 0;
for i in range(0, 1024):
    if eval(x, y):
        sum += 1

print(sum / 1024.0)
```

脚本里的 1024 表示给这个电路1024次输入，将这1024次输入累加起来再除以1024，模拟了随机计算的 stochastic to conventional converter.