Modifications of statistics under dimer diffusion
Supplementary materials

Georges Sitja
CNRS, UMR 7325, Aix-Marseille University, Cinam, Campus Luminy, Case 913,
F-13288 Marseille 09, France.

1 Appendix 4 - Proofs

Proofs of the validity of various tricky formulas.

1.1 Validation of the formula (38) for $P_2(x)$

We have obtained:

$$P_2(x) = \frac{A+P_2}{A^2}(A-x)^2 - (A-x)$$

and it is good to check that it fulfills the differential equation (35)

$$\begin{aligned}
P_2'(x) &= \frac{A+P_2}{A^2}[-2(A-x)] + 1 \\
&= -2\frac{A+P_2}{A^2}(A-x) + 2 - 1 \\
&= \frac{1}{A-x}\left[-2\frac{A+P_2}{A^2}(A-x)^2 + 2(A-x)\right] - 1 \\
&= -\frac{2}{A-x}\left[\frac{A+P_2}{A^2}(A-x)^2 - (A-x)\right] - 1 \\
&= -\frac{2}{A-x}P_2(x) - 1
\end{aligned}$$

That establish the validity of the differential equation.

1.2 Verification of the formula (42) for $P_3(x)$

Let's start by expanding P_3 to separate the different forms of contributions of x:

$$P_3(x) = \left[\frac{P_3+2A}{A^2} + 2\frac{A+P_2}{A^2}(\ ln(A) - ln(A-x)\ )\right](A-x)^2 - 2(A-x)$$

$$= \left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A) - 2\frac{A+P_2}{A^2}ln(A-x)\right](A-x)^2 - 2(A-x)$$

$$= \left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x)^2 - 2\frac{A+P_2}{A^2}ln(A-x)(A-x)^2 - 2(A-x)$$

The following derivation and reorganization steps:

$$P_3'(x) = -2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) - 2\frac{A+P_2}{A^2}\left[-\frac{(A-x)^2}{(A-x)} - 2ln(A-x)(A-x)\right] + 2$$

$$= -2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) + 2\frac{A+P_2}{A^2}\left[(A-x) + 2ln(A-x)(A-x)\right] + 2$$

$$= -2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) + 2\frac{A+P_2}{A^2}(A-x) + 4\frac{A+P_2}{A^2}ln(A-x)(A-x) + 2$$

$$= 2\frac{A+P_2}{A^2}(A-x) - 2 - 2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) + 4\frac{A+P_2}{A^2}ln(A-x)(A-x) + 2 + 2$$

$$= \frac{2}{A-x}\left[\frac{A+P_2}{A^2}(A-x)^2 - (A-x)\right] - 2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) + 4\frac{A+P_2}{A^2}ln(A-x)(A-x) + 4$$

$$= \frac{2}{A-x}P_2(x) - 2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A)\right](A-x) + 4\frac{A+P_2}{A^2}ln(A-x)(A-x) + 4$$

$$= \frac{2}{A-x}P_2(x) - 2\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}ln(A) - 2\frac{A+P_2}{A^2}ln(A-x)\right](A-x) + 4$$

$$= \frac{2}{A-x}P_2(x) - 2\left\{\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}[\ ln(A) - ln(A-x)\ ]\right](A-x) - 2\right\}$$

$$= \frac{2}{A-x}P_2(x) - \frac{2}{A-x}\left\{\left[\frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2}[\ ln(A) - ln(A-x)\ ]\right](A-x)^2 - 2(A-x)\right\}$$

$$= \frac{2}{A-x}P_2(x) - \frac{2}{A-x}P_3(x)$$

definitively demonstrate the validity of the formula (42).

1.3 Validation of the formula (47) for $P_{n\geq 3}(x)$

Let's start with the generic formula for $f_n(x)$ and let's derivate it to check that we indeed obtain $\frac{2}{A-x}f_{n-1}(x)$. As the generic formula (46) shows us:

$$f_{(n\geq 3)}(x) = K_n + \left[\sum_{i=1}^{n-2} K_{n-i}\frac{(-2)^i}{i!}[\ ln(A-x)\ ]^i\right] - \frac{2^{(n-2)}}{A-x}$$

$$f'_{(n\geq 3)}(x) = \left[\sum_{i=1}^{n-2} K_{n-i}\frac{(-2)^i}{i!}\ (-i)\frac{[ln(A-x)\ ]^{i-1}}{A-x}\right] - \frac{2^{(n-2)}}{(A-x)^2}$$

$$= \left[\sum_{i=1}^{n-2} K_{n-i}\frac{(-2)(-2)^{i-1}}{i!}\ (-i)\frac{[ln(A-x)\ ]^{i-1}}{A-x}\right] - \frac{2}{A-x}\left[\frac{2^{(n-2-1)}}{(A-x)}\right]$$

$$= \left[\sum_{i=1}^{n-2} K_{n-i}\frac{(2)(-2)^{i-1}}{i!}(i)\ \frac{[ln(A-x)\ ]^{i-1}}{A-x}\right] + \frac{2}{A-x}\left[-\frac{2^{(n-2-1)}}{(A-x)}\right]$$

$$= \left[\sum_{i=1}^{n-2} K_{n-i}\frac{2}{A-x}\frac{(-2)^{i-1}}{(i-1)!}\ [ln(A-x)\ ]^{i-1}\right] + \frac{2}{A-x}\left[-\frac{2^{(n-2-1)}}{(A-x)}\right]$$

$$= \frac{2}{A-x}\left[\sum_{i=1}^{n-2} K_{n-i}\frac{(-2)^{i-1}}{(i-1)!}\ [ln(A-x)\ ]^{i-1}\right] + \frac{2}{A-x}\left[-\frac{2^{(n-2-1)}}{(A-x)}\right]$$

$$= \frac{2}{A-x}\left[\sum_{i=0}^{n-3} K_{n-1-i}\frac{(-2)^i}{i!}\ [ln(A-x)\ ]^i\right] + \frac{2}{A-x}\left[-\frac{2^{(n-1-2)}}{(A-x)}\right]$$

$$= \frac{2}{A-x}\left[K_{n-1-0}\frac{(-2)^0}{0!}\ [ln(A-x)\ ]^0 + \sum_{i=1}^{n-1-2} K_{n-1-i}\frac{(-2)^i}{i!}\ [ln(A-x)\ ]^i\right] + \frac{2}{A-x}\left[-\frac{2^{(n-1)-2}}{(A-x)}\right]$$

$$= \frac{2}{A-x}\left[K_{n-1} + \sum_{i=1}^{(n-1)-2} K_{(n-1)-i}\frac{(-2)^i}{i!}\ [ln(A-x)\ ]^i\right] + \frac{2}{A-x}\left[-\frac{2^{(n-1)-2}}{(A-x)}\right]$$

$$= \frac{2}{A-x}f_{n-1}(x)$$

The recurrence relation is verified.

We have furthermore:

$$f_3(x) = K_3 + \left[\sum_{i=1}^{3-2} K_{3-i}\frac{(-2)^i}{i!}[\ ln(A-x)\ ]^i\right] - \frac{2^{(3-2)}}{A-x}$$

$$= K_3 + \left[K_{3-1}\frac{(-2)^1}{1!}[\ ln(A-x)\ ]^1\right] - \frac{2^1}{A-x}$$

$$= K_3 + [K_2(-2)[\ ln(A-x)\ ]] - \frac{2}{A-x}$$

$$= K_3 - 2K_2 ln(A-x)\ -\ \frac{2}{A-x}$$

which gives for P_3:

$$P_3(x) = f_3(x)(A-x)^2 = K_3(A-x)^2 - 2K_2 ln(A-x)(A-x)^2 \ -2(A-x)$$

This is consistent with the formula for P_3 (43) obtained earlier with

$$K_2 = \frac{A+P_2}{A^2} \qquad and \qquad K_3 \ = \frac{P_3+2A}{A^2} + 2\frac{(A+P_2)}{A^2} ln(A)$$

The formula (47) is then fully prooved by induction.

2 Appendix 5 - Useful computer programs

Here, to assist in the calculation of the final size distributions for the different cases, I include several programs written in python. As the programs are in python, you will have to respect the indentation to make them work. To limit the risk of error, I will give usage examples and the obtained outputs.

2.1 Monomer diffusion calculation

The program below allows to calculate the statistics after the diffusion of the monomers.

Usage and output examples:

```
--------------------------------------------------------------------------------
Command line :

./mdiff.py 0.4 0.3 0.2 0.1 0 0 0 0


Output :

There are  8 probabilities.
The process stops after  0.5  movements.
P 0  :  0.4  ->  0.63608160417242
P 1  :  0.3  ->  0.0
P 2  :  0.2  ->  0.16679593142097418
P 3  :  0.1  ->  0.13646939843534253
P 4  :  0.0  ->  0.04833291194585049
P 5  :  0.0  ->  0.010487925990864287
P 6  :  0.0  ->  0.0016189945994933574
P 7  :  0.0  ->  0.0001929254888594872
P 8  :  -2.7755575615628914e-17  ->  1.8674378764464476e-05
P 9  :  0.0  ->  1.519962022357588e-06
P 10  :  0.0  ->  1.0665197437654128e-07

Evolution of occupied sites :  0.6  ->  0.36391839582758

Sum of probabilities :  0.9999999930465655
--------------------------------------------------------------------------------
Command line :

./mdiff.py 0.4 0.3 0.2


Output :

There are  3 probabilities.
The process stops after  0.5  movements.
P 0  :  0.4  ->  0.63608160417242
P 1  :  0.3  ->  0.0
P 2  :  0.2  ->  0.16679593142097418
P 3  :  0.09999999999999998  ->  0.1364693984353425
P 4  :  0.0  ->  0.048332911945850474
P 5  :  0.0  ->  0.010487925990864285

Evolution of occupied sites :  0.6  ->  0.36391839582758

Sum of probabilities :  0.9981677719654513
--------------------------------------------------------------------------------
Command line :

./mdiff.py 0.4 0.3  -line


Output :

0.63608160417242  0.0  0.2274489973922375  0.10614286544971084  0.02558801220662672
 --------------------------------------------------------------------------------
```

If we give N probabilities, the program calculates the N+1th probability to make the sum equal to 1. In output we will have n+3 values.
The probabilities must be given in order, starting with P_0.
The program sums the calculated probabilities to check that we obtain 1 (or close to 1 if we did not provide enough probabilities). By adding the "-line" option, the output of the program is simplified: it is simply the list of the final probabilities on a line, in order to be able to easily launch the calculation of the dimer diffusion for which the program is provided below.

mdiff.py:

```
------------------------------------------------------------------------------
#!/usr/bin/python
import math
import sys

concat=0;

nbp=len(sys.argv) - 1;

p=(nbp+4)*[0.0];
jp=(nbp+4)*[0.0];
reste=1.0;

for i in range(0,nbp) :
   if str(sys.argv[i+1]) == "-line" :
      concat = 1;
      i=i+1;
   else :
      index=i+1;
      p[i-concat]=float(sys.argv[index]);
      reste=reste - p[i-concat];

nbp = nbp-concat ;
if concat == 0 :
   print ("There are ", nbp, "probabilities.");
p[nbp]=reste;


xa=p[1] / (1 - p[0]);
if concat == 0 :
   print ("The process stops after ",xa," movements.");

jp[0]=(p[0]-1)*math.exp(-xa)  + 1;
for s in range(1,nbp+3) :
   jp[s]=0.0
   for i in range(0,s) : # en fait la dernière boucle c'est s-1, c'est ce qu'on veut !
       jp[s]=jp[s]+xa**i*p[s-i]/math.factorial(i);
   jp[s]=jp[s]+1/math.factorial(s)*(p[0] - 1)*(xa**s);
   jp[s]=jp[s]*math.exp(-xa);

somme=0.0;
for i in range(0,nbp+3) :
   somme=somme+jp[i];
   if concat == 0 :
      print("P",i," : ",p[i]," -> ",jp[i]);
   else :
      print(jp[i]," ",end="");

if concat == 1 :
   print("")


occupej=1-jp[0];
occupe=1-p[0];

if concat == 0 :
   print("")
   print("Evolution of occupied sites : ",occupe," -> ",occupej);
   print("");
```

```
   print("Sum of probabilities : ",somme)
   print("")
--------------------------------------------------------------------------------
```

2.2 Dimer diffusion calculation

The input parameters for the following programs are the same as for the program calculating the diffusion of the monomers, i.e. the list in order of the initial probabilities, starting with P_0. The "-line" option does not exist here, as I will not study the trimer "diffusion".

Usage and output example:

```
--------------------------------------------------------------------------------
Command line :

./ddiff.py 0.5 0 0.4 0.1  0


Output :

There are  5 probabilities.

The process stops after  0.8  movements.

P 0  :  0.5  ->  0.7753355179413892
P 1  :  0.0  ->  0.0
P 2  :  0.4  ->  0.0
P 3  :  0.1  ->  0.044932896411722156
P 4  :  0.0  ->  0.07189263425875546
P 5  :  -2.7755575615628914e-17  ->  0.03594631712937772
P 6  :  0.0  ->  0.03834273827133625
P 7  :  0.0  ->  0.014378526851751084

Evolution of occupied sites :  0.5  ->  0.22466448205861078

Sum of probabilities :  0.9808286308643319
--------------------------------------------------------------------------------
```

ddiff.py:

```
--------------------------------------------------------------------------------
#!/usr/bin/python
# calcule les nouvesses probabilités en cas de diffusion des dimères en "bloc"
import math
import sys

nbp=len(sys.argv) - 1; #Le nombre de probabilités

p=(nbp+4)*[0.0]; # les probas initiales
jp=(nbp+4)*[0.0]; # les probas finales
p[nbp]=1.0;


print ("There are ", nbp, "probabilities.")
for i in range(0,nbp) :
   p[i]=float(sys.argv[i+1]) # on assigne les probabiltés données dans le tableau
   p[nbp]=p[nbp] - p[i]; # on fait en sorte que la somme des probas soit 1


xa=p[2] / (1 - p[0]);
print("");
print ("The process stops after ",xa," movements.");

jp[0]=1 - (1 - p[0])*math.exp(-xa);

s=1;
while s <= nbp+2:
   #--------------------s impair----------------------------
```

```
   jp[s]=0.0;
   k=0
   while k <= (s-1)//2 :
       jp[s]=jp[s]+xa**k*p[s-2*k]/math.factorial(k);
       k=k+1;
   jp[s]=jp[s]*math.exp(-xa);
   #--------------------s impair----------------------------

   s=s+1;
   #---------------------s pair-----------------------------
   jp[s]=0.0;
   k=0
   while k <= s//2-1 :
       jp[s]=jp[s]+xa**k*p[s-2*k]/math.factorial(k);
       k=k+1;
   jp[s]=jp[s]+1/math.factorial(s//2)*(p[0] - 1)*(xa**(s/2));
   jp[s]=jp[s]*math.exp(-xa);
   #---------------------s pair-----------------------------
   s=s+1
print("")
somme=0.0;
for i in range(0,nbp+3) :
   somme=somme+jp[i];
   print("P",i," : ",p[i]," -> ",jp[i]);

occupej=1-jp[0];
occupe=1-p[0];

print("")

print("Evolution of occupied sites : ",occupe," -> ",occupej);
print("");
print("Sum of probabilities : ",somme)
print("")
-------------------------------------------------------------------------------
```

2.3 Dimer explosion calculation

The input parameters for the following programs are the same as for the program calculating the diffusion of the monomers, i.e. the list in order of the initial probabilities, starting with P_0. The "-line" option does not exist here.

Usage and output example:

```
-------------------------------------------------------------------------------
Command line :

./dexplos.py 0.5 0 0.4 0.1  0


Output :

There are  5 probabilities.

The process stops after  0.2222222222222222  movements.

P 0  :  0.5  ->  0.7222222222222222
P 1  :  0.0  ->  0.0
P 2  :  0.4  ->  0.0
P 3  :  0.1  ->  0.11049876445179463
P 4  :  0.0  ->  0.09722788795395765
P 5  :  -2.7755575615628914e-17  ->  0.047815786515095215
P 6  :  0.0  ->  0.016583484825492434
P 7  :  0.0  ->  0.0044574530442513804

Evolution of occupied sites :  0.5  ->  0.2777777777777778
Sum of probabilities :  0.9988055990128135
-------------------------------------------------------------------------------
```

dexplos.py:

```
--------------------------------------------------------------------------------
#!/usr/bin/python
# calcule les nouvesses probabilités en cas d'explosion des dimères
import math
import sys

nbp=len(sys.argv) - 1; #Le nombre de probabilités

p=(nbp+4)*[0.0]; # les probas initiales
jp=(nbp+4)*[0.0]; # les probas finales
jK=(nbp+4)*[0.0]; # les K_n
p[nbp]=1.0;


print ("There are ", nbp, "probabilities.")
for i in range(0,nbp) :
   p[i]=float(sys.argv[i+1]) # on assigne les probabiltés données dans le tableau
   p[nbp]=p[nbp] - p[i]; # on fait en sorte que la somme des probas soit 1

A=1-p[0];
xa=A - A**2/(A+p[2]);

print("");
print ("The process stops after ",xa," movements.");

jp[0]=p[0]+xa;
jp[1]=0.0;
jp[2]=0.0;

jK[2] = (A + p[2])/A**2;
LNA=-2*math.log(A);
#-------------------------Calcul des constantes-----------------------
s=3;
while s <= nbp+2:
   jK[s]=( p[s] + 2**(s-2)*A ) / A**2;
   i=1
   while i <= s-2:
      jK[s] = jK[s] - jK[s-i]*(LNA**i)/math.factorial(i);
      i=i+1;
   s=s+1

#-------------------------Calcul des constantes-----------------------

LNA=-2*math.log(A-xa);
#-------------------------Calcul des probas----------------------
s=3;
while s <= nbp+2:
   jp[s] = jK[s];
   i=1
   while i <= s-2:
      jp[s] = jp[s] + jK[s-i]*(LNA**i)/math.factorial(i);
      i=i+1;
   jp[s] = jp[s]*((A-xa)**2) - (2**(s-2))*(A-xa);
   s=s+1;
#-------------------------Calcul des probas----------------------

print("")
somme=0.0
for i in range(0,nbp+3) :
   print("P",i," : ",p[i]," -> ",jp[i]);
   somme=somme+jp[i];

occupej=1-jp[0];
occupe=1-p[0];

print("")

print("Evolution of occupied sites : ",occupe," -> ",occupej);

print("Sum of probabilities : ",somme)
print("");
```

2.4 Dimer evaporation calculation

The input parameters for the following programs are the same as for the program calculating the diffusion of the monomers, i.e. the list in order of the initial probabilities, starting with P_0. The "-line" option does not exist here.

Usage and output example:

```
------------------------------------------------------------------------------

Command line :

./devap.py 0.5 0 0.4 0.1  0


Output :

There are  5 probabilities.
The process stops after  0.4508066615170332  movements.

P 0  :  0.5  ->  0.3185570044635628
P 1  :  0.0  ->  0.5064936107579598
P 2  :  0.4  ->  3.5366932101106616e-17
P 3  :  0.1  ->  0.11872237419187774
P 4  :  0.0  ->  0.04543713198682613
P 5  :  -2.7755575615628914e-17  ->  0.009318277295881153
P 6  :  0.0  ->  0.0013162491387744963
P 7  :  0.0  ->  0.00014199246540335273

Evolution of occupied sites :   0.5  ->  0.6814429955364372

Sum of probabilities :  0.9999866403002855

------------------------------------------------------------------------------
```

devap.py:

```
-------------------------------------------------------------------------------
#!/usr/bin/python
# calcule les nouvesses probabilités en cas d'évaporation
import math
import sys

nbp=len(sys.argv) - 1; #Le nombre de probabilités

p=(nbp+4)*[0.0]; # les probas initiales
jp=(nbp+4)*[0.0]; # les probas finales
p[nbp]=1.0;

print ("There are ", nbp, "probabilities.")
for i in range(0,nbp) :
   p[i]=float(sys.argv[i+1]) # on assigne les probabiltés données dans le tableau
   p[nbp]=p[nbp] - p[i]; # on fait en sorte que la somme des probas soit 1

a=(1-p[1]);
b=2*p[0]*p[2];

if p[0] == 0 :
   xa=p[2] / a ;
else :
   xa=(a - (a*a - b)**(1/2))/p[0];

print ("The process stops after ",xa," movements.");

jp[0]=p[0]*math.exp(-xa);
jp[1]=1-(1-p[1]-xa*p[0])*math.exp(-xa);

s=2;
while s <= nbp+2:
   jp[s]=0.0;
```

```
   k=0
   while k <= s :
       jp[s]=jp[s]+p[s-k] * (xa**k) / math.factorial(k);
       k=k+1;
   jp[s]=jp[s] - (xa**(s-1)) / math.factorial(s-1);
   jp[s]=jp[s]*math.exp(-xa);
   s=s+1;

somme=0.0;
print("")
for i in range(0,nbp+3) :
   somme=somme+jp[i];
   print("P",i," : ",p[i]," -> ",jp[i]);

occupej=1-jp[0];
occupe=1-p[0];

print("")

print("Evolution of occupied sites :  ",occupe," -> ",occupej);
print("")
print("Sum of probabilities : ",somme)
print("")
--------------------------------------------------------------------------------
```