(%o33) 3

(%o34) 5

(%o35) 1

(%o36) 0

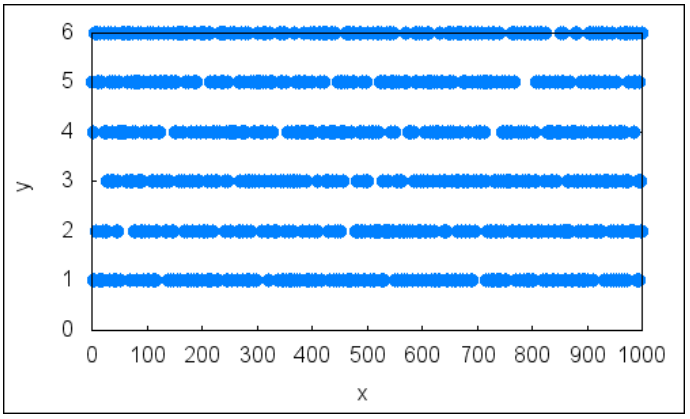
(%o37) 4

(%o38) 3

Liste de simulation du lancer de dé

```
(%i39) Liste:[]$
       N:1000$
       for i from 1 thru N do Liste:append(Liste,[random(6)+1])$
       Liste$
       entiers:makelist(n,n,1,N)$
       wxplot2d([discrete, entiers, Liste], [style, points], [y,0,6])$
```

(%t44)



Transformation d'une liste en un ensemble

```
(%i45) set:setify(Liste);
       for s in set do print (s);
```

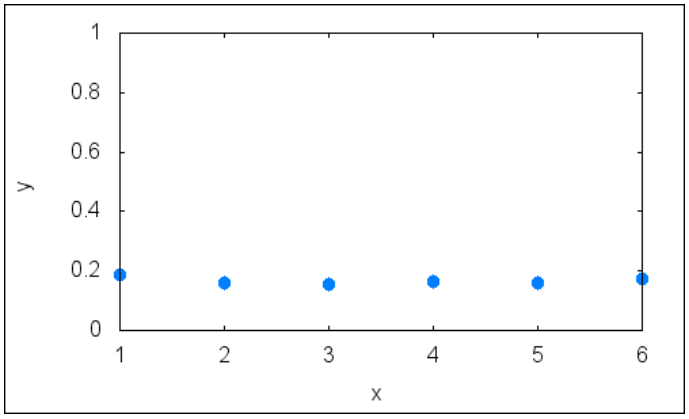
(%o45) $1, 2, 3, 4, 5, 6123456$

(%o46) *done*

Calcul des fréquences

```
(%i47) listeFrequences:[]$
       for element in set do (
           s:0,
           for i step 1 from 1 thru length(Liste) do (
               if element=Liste[i] then s:s+1
           ),
           listeFrequences:append(listeFrequences, [s/length(Liste)])
       )$
       float(listeFrequences);
       entiers:makelist(n,n,1,6)$
       wxplot2d([discrete, entiers, listeFrequences], [style, points], [y,0,1]);
```

(%o49) $[0.185, 0.159, 0.157, 0.165, 0.162, 0.172]$



(%t51)

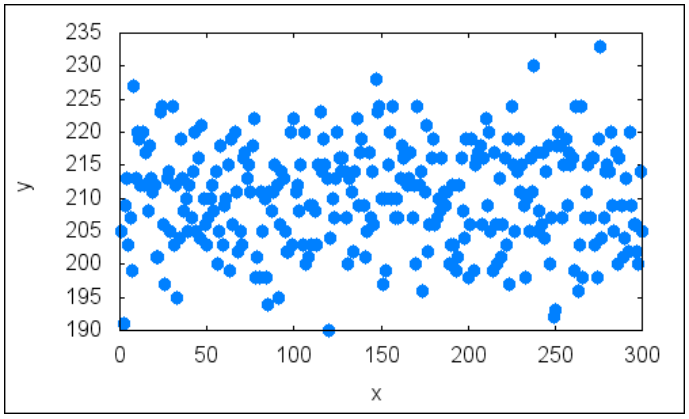
(%o51)

9.1.3 Simulation de la loi binomiale

```
(%i52) nombreSucces(N,p):=block([],
           somme:0,
           for i from 1 thru N do (
               somme:somme+floor(random(1.0)+p)
               ),
           somme
       )$

(%i53) nombreSucces(30,0.7);
```

(%o53) 26

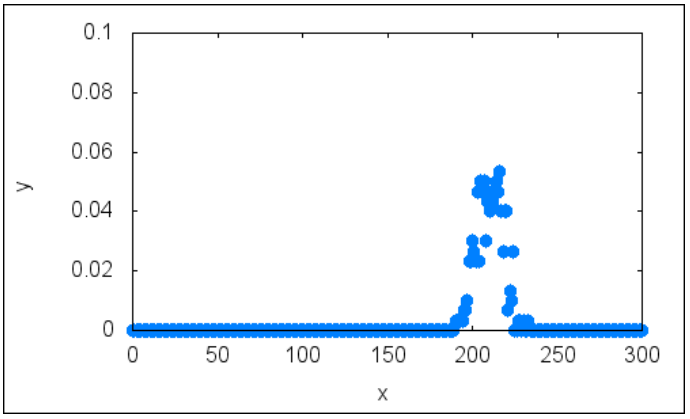
```
(%i54) Liste:[]$
       N:300$
       p:0.7$
       for i from 1 thru N do Liste:append(Liste,[nombreSucces(N,p)])$
       entiers:makelist(n,n,1,N)$
       wxplot2d([discrete, entiers, Liste], [style, points]);
```



(%t59)

(%o59)

```
(%i60) listeFrequences:[]$
       listeValeurs:makelist(i,i,0,N)$
       for element in setify(listeValeurs) do (
           s:0,
           for i step 1 from 1 thru length(Liste) do (
               if element=Liste[i] then s:s+1
           ),
           listeFrequences:append(listeFrequences, [s/length(Liste)])
       )$
       wxplot2d([discrete, listeValeurs, listeFrequences], [style, points],[y,0,0.1]);
```



(%t63)

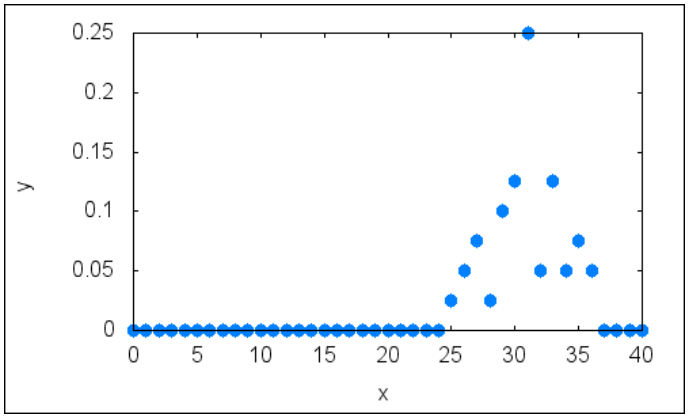
(%o63)

Définition de la fonction simulation de la loi binomiale, elle retourne une liste :

```
(%i64) declare(N, integer)$
       assume(N>0)$
       simulationLoiBinomiale(N,p):=block([],
           Liste:[],
           for i from 1 thru N do Liste:append(Liste,[nombreSucces(N,p)]),
           listeFrequences:[],
           listeValeurs:makelist(i,i,0,N),
           for element in setify(listeValeurs) do (
               s:0,
               for i step 1 from 1 thru length(Liste) do (
                   if element=Liste[i] then s:s+1
                   ),
               listeFrequences:append(listeFrequences, [s/length(Liste)])
               ),
           listeFrequences
       )$
```

Définition de la fonction affichage d'une simulation de la loi binomiale :

```
(%i67) afficheSimulationLoiBinomiale(N,p):=block([],
           listeFrequences:simulationLoiBinomiale(N,p),
           wxplot2d([discrete, makelist(i,i,0,N), listeFrequences], [style, points])
       )$

(%i68) afficheSimulationLoiBinomiale(40,0.8);
```



(%t68)

(%o68)

9.1.4 Simulation de la loi uniforme

Tirage aléatoire d'un nombre de l'intervalle $[0;1[$

```
(%i69) random(1.0);
```

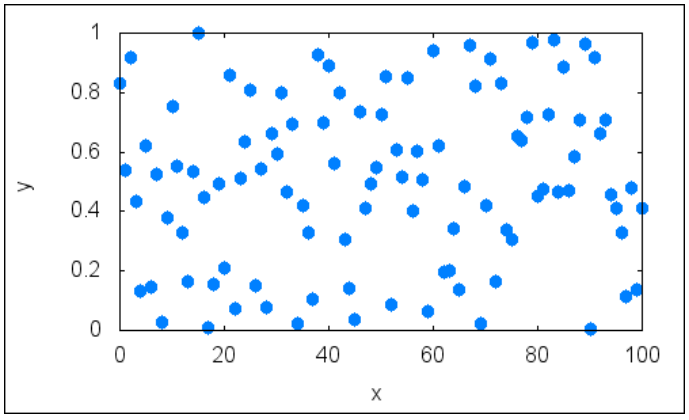
(%o69) 0.95659153069339

Création d'une liste de nombres aléatoires de l'intervalle $[0;1[$

```
(%i70) N:100$
       Liste:[]$
       for i from 0 thru N do Liste:append(Liste,[random(1.0)])$
       Liste$
```

Affichage de la liste précédente

```
(%i74) entiers:makelist(n,n,0,N)$
       wxplot2d([discrete, entiers, Liste], [style, points],[y, 0, 1]);
```



(%t75)

(%o75)

9.2 Distributions

```
(%i76) load(distrib)$
```

9.2.1 Schéma de Bernoulli

À retenir :

pdf : probability density function, c'est la fonction densité de probabilité

cdf : cumulative density function, c'est la fonction de répartition pdf_bernoulli(1,p) donne la valeur de $P(X = 1)$ où X suit la loi de Bernoulli de paramètre p :

```
(%i77) assume(0<p,p<1)$
       pdf_bernoulli(1,p);
       pdf_bernoulli(0,p);
```

(%o78) 0.7

(%o79) 0.3

Représentation graphique

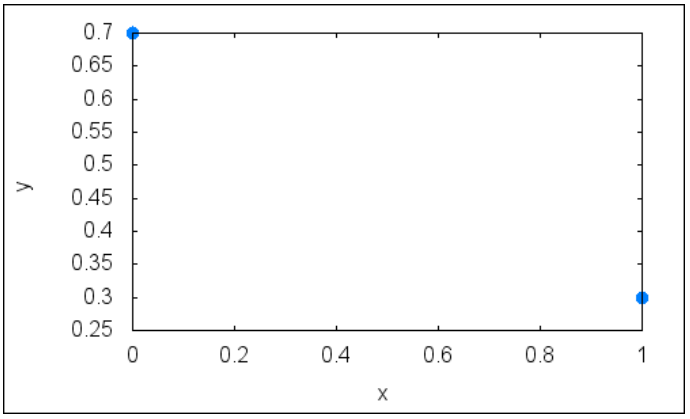
```
(%i80) p:0.3;
       zeroUn:[0, 1]$
       f1:p$
       f0:1-f1$
       float(f0);
       float(f1);
       ListeFrequences:[f0, f1]$
       wxplot2d([discrete, zeroUn, ListeFrequences], [style, points]);
```

(%o80) 0.3

(%o84) 0.7

(%o85) 0.3

(%t87)



(%o87)

9.2.2 Loi binomiale

pdf_binomial(k,n,p) ; donne la valeur de $P(X = k)$ où X suit la loi binomiale de paramètres n et p :

```
(%i88) pdf_binomial(5,7,1/6);
```

(%o88) $\frac{175}{93312}$

cdf_binomial(k,n,p) ; donne la valeur de $P(X \leq k)$ où X suit la loi binomiale de paramètres n et p :

```
(%i89) cdf_binomial(5,7,1/6);
```

(%o89) $\frac{7775}{7776}$

Calcul de $P(a \leq X \leq b)$

```
(%i90) n:20$
       p:0.3$
       a:2$
       b:5$
       cdf_binomial(b,n,p)-cdf_binomial(a-1,n,p);
```

(%o94) 0.40873356967328

Coefficient binomial

```
(%i95) binomial (5, 2);
```

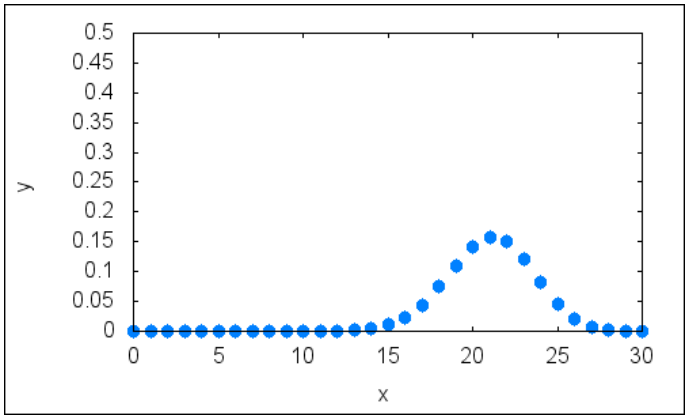
(%o95) 10

Représentation graphique de la loi binomiale

```
(%i96) N:30;
       entiers:makelist(n,n,0,N)$
       listeBinomiale:makelist(pdf_binomial(n,N,0.7),n,0,N)$
       wxplot2d([discrete, entiers, listeBinomiale], [style, points], [y,0,0.5]);
```

(%o96) 30

(%t99)

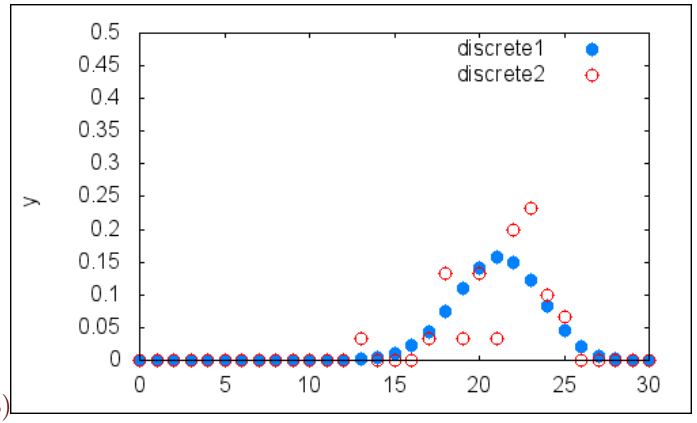


(%o99)

Comparaison avec une simulation

```
(%i100)N:30;
       entiers:makelist(n,n,0,N)$
       listeBinomiale:makelist(pdf_binomial(n,N,0.7),n,0,N)$
       wxplot2d([[discrete, entiers, listeBinomiale], [discrete, entiers, simulationLoiBinomial
```

(%o100)30



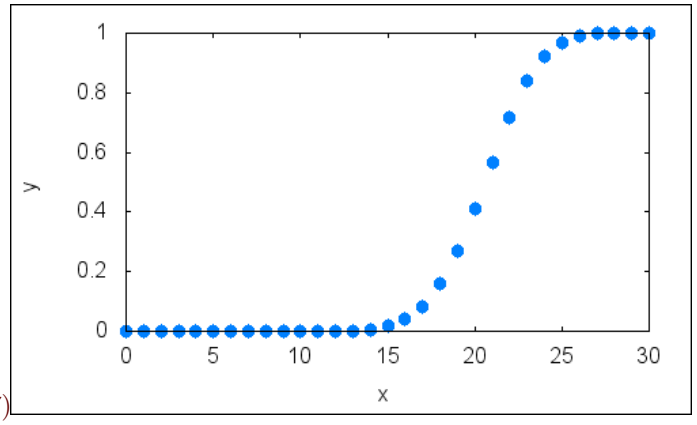
(%t103)

(%o103)

Représentation graphique de la fonction de répartition de loi binomiale

```
(%i104)N:30;
       entiers:makelist(n,n,0,N)$
       listeBinomiale:makelist(cdf_binomial(n,N,0.7),n,0,N)$
       wxplot2d([discrete, entiers, listeBinomiale], [style, points], [y,0,1]);
```

(%o104)30



(%t107)

(%o107)

9.2.3 Loi de Poisson

float(pdf_poisson(k,lambda)) ; donne une valeur approchée de $P(X = k) = \frac{\lambda^k}{k!}e^(-\lambda)$:

```
(%i108)float(pdf_poisson(3,95/100));
```

(%o108)0.055263680830717

float(cdf_poisson(k,lambda)) ; donne une valeur approchée de $P(X \leq k)$:

```
(%i109)float(cdf_poisson(3,95/100));
```

(%o109)0.98392556340084

Calcul de $P(a \leq X \leq b)$

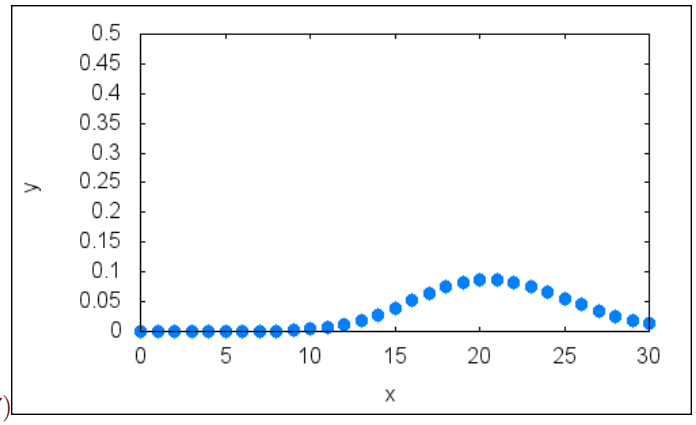
```
(%i110)lambda:7.3$
       a:2$
       b:5$
       cdf_poisson(b,lambda)-cdf_poisson(a-1,lambda);
```

(%o113)0.258436232947

Représentation graphique de la loi de Poisson

```
(%i114)N:30;
       entiers:makelist(n,n,0,N)$
       listePoisson:makelist(pdf_poisson(n,21),n,0,N)$
       wxplot2d([discrete, entiers, listePoisson], [style, points], [y,0,0.5]);
```

(%o114)30



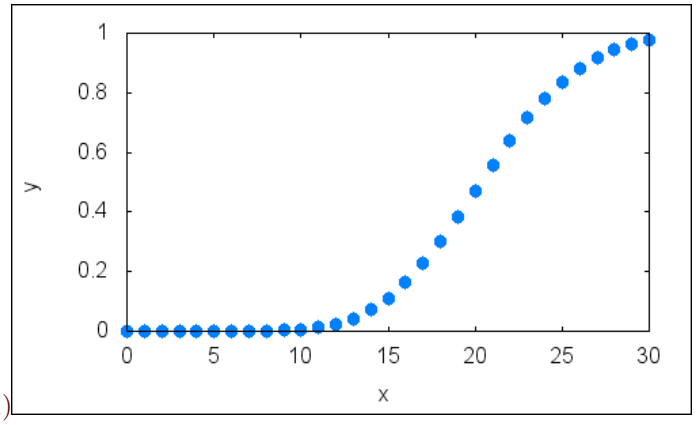
(%t117)

(%o117)

Représentation graphique de la fonction de répartition de loi de Poisson

```
(%i118)N:30;
       entiers:makelist(n,n,0,N)$
       listePoisson:makelist(cdf_poisson(n,21),n,0,N)$
       wxplot2d([discrete, entiers, listePoisson], [style, points], [y,0,1]);
```

(%o118)30



(%t121)

(%o121)

9.2.4 Loi normale

float(pdf_normal(k,m,s)) ; donne une valeur approchée de la fonction densité de la loi normale d'espérance m et d'écart-type s :

```
(%i122)float(pdf_normal(95/100,0,1));
```

(%o122)0.25405905646919

float(cdf_normal(k,m,s)) ; donne une valeur approchée de la fonction répartition de la loi normale d'espérance m et d'écart-type s :

```
(%i123)float(cdf_normal(95/100,0,1));
```

(%o123)0.82894387369152

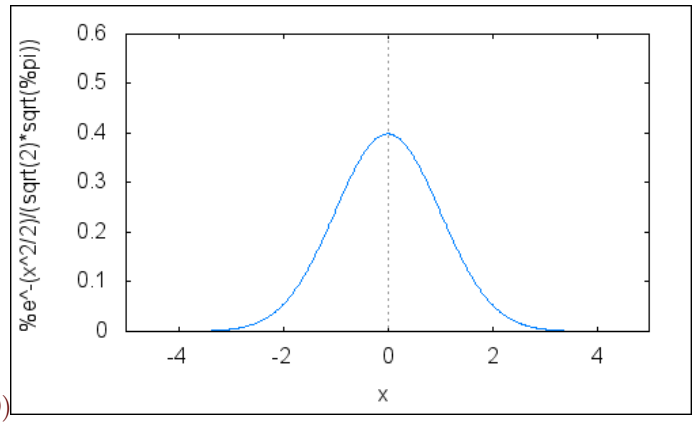
Calcul de $P(a \leq X \leq b)$

```
(%i124)m:0.95$
       s:1.5$
       a:2$
       b:5$
       float(cdf_normal(b,m,s)-cdf_normal(a,m,s));
```

(%o128)0.23849667842003

Représentation graphique de la loi normale Loi normale centrée réduite

```
(%i129)wxplot2d(pdf_normal(x,0,1),[x,-5,5], [y,0,0.6]);
```

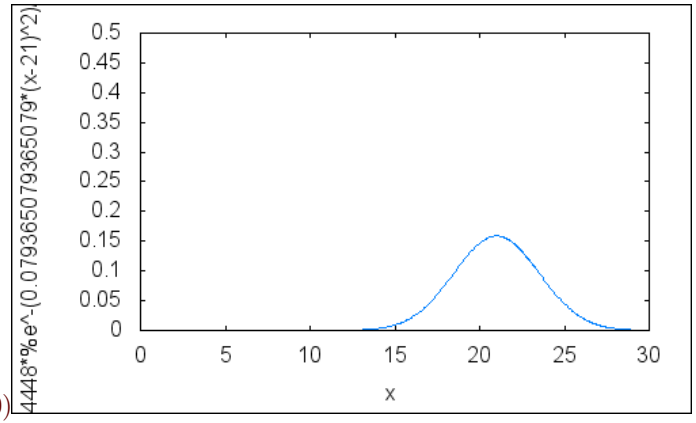


(%t129)

(%o129)

Loi normale de paramètres 21 et 6.3 (variance)

```
(%i130)wxplot2d(pdf_normal(x,21,sqrt(6.3)),[x,0,30], [y,0,0.5]);
```

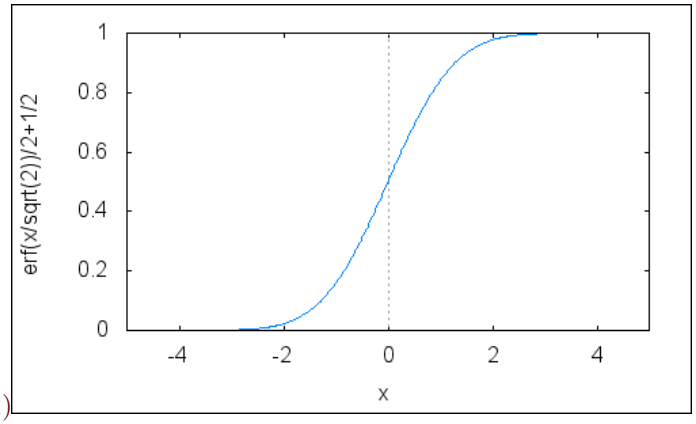


(%t130)

(%o130)

Représentation graphique de la fonction de répartition de la loi normale Fonction de répartition de la loi normale centrée réduite

```
(%i131) wxplot2d(cdf_normal(x,0,1),[x,-5,5], [y,0,1]);
```

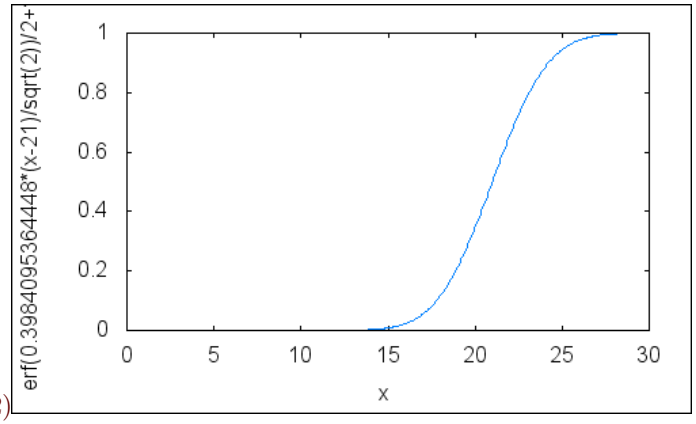


(%t131)

(%o131)

F

```
(%i132) wxplot2d(cdf_normal(x,21,sqrt(6.3)),[x,0,30], [y,0,1]);
```



(%t132)

(%o132)

9.2.5 Approximation d'une loi binomiale par une loi normale

```
(%i133)N:30;
       p:0.7;
       N*p;
       N*(1-p);
       entiers:makelist(n,n,0,N)$
       listeBinomiale:makelist(pdf_binomial(n,N,p),n,0,N)$
       wxplot2d([[discrete, entiers, listeBinomiale], pdf_normal(x,N*p,sqrt(N*p*(1-p)))],
                   [x,0,N],
                   [style, points, lines],
                   [y,0,0.2],
                   [legend, "Loi binomiale", "Loi normale"],
                   [xlabel, "x"],


                   [ylabel, "y"]
       );
```

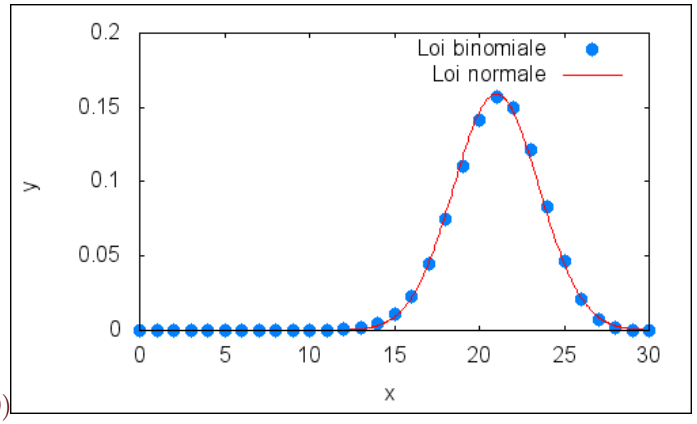
(%o133) 30

(%o134) 0.7

(%o135) 21.0

(%o136) 9.000000000000002

(%t139)



(%o139)

9.2.6 Approximation d'une loi binomiale par une loi de Poisson

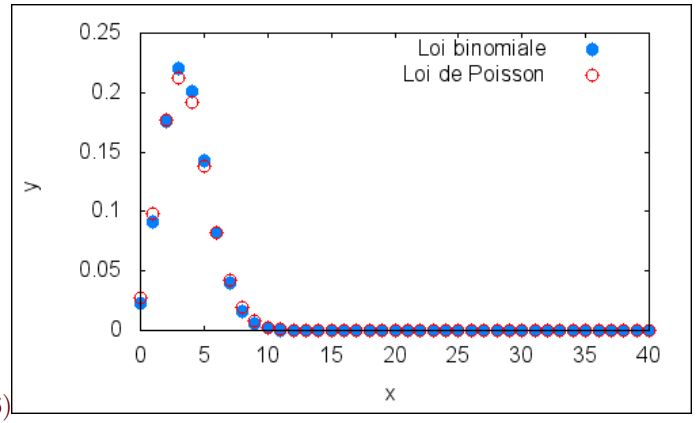
```
(%i140) N:40;
        p:0.09;
        N*p*(1-p);
        entiers:makelist(n,n,0,N)$
        listeBinomiale:makelist(pdf_binomial(n,N,p),n,0,N)$
        listePoisson:makelist(pdf_poisson(n,N*p),n,0,N)$
        wxplot2d([[discrete, entiers, listeBinomiale], [discrete, entiers, listePoisson]],
                    [x,0,N],
                    [style, points, points],
                    [y,0,0.25],
                    [legend, "Loi binomiale", "Loi de Poisson"],
                    [xlabel, "x"],


                    [ylabel, "y"]
        );
```

(%o140) 40

(%o141) 0.09

(%o142) 3.276

(%t146)



(%o146)

Chapitre 10

Algorithmique et programmation

10.1 Boucle Tant que

Boucle << tant que >> (<< while >>)

```
(%i1)  n:0$
       s:0$
       while n < 10 do (
           n : n + 1,
           s : s + n
       )$
       s;
```

(%o4) 55

10.2 Boucle Pour

Boucle << pour >> (<< for >>), le pas est égal à 1 par défaut :

```
(%i5)  s:0$
       for n from 0 thru 10 do (
           s : s + n
       )$
       s;
```

(%o7) 55

On peut écrire la ligne for en une seule ligne :

```
(%i8)  s:0$
       for n from 0 thru 10 do s : s + n$
       s;
```

(%o10) 55

Avec un pas de 2

```
(%i11) s:0$
       for n step 2 from 0 thru 10 do s : s + n$
       s;
```

(%o13) 30

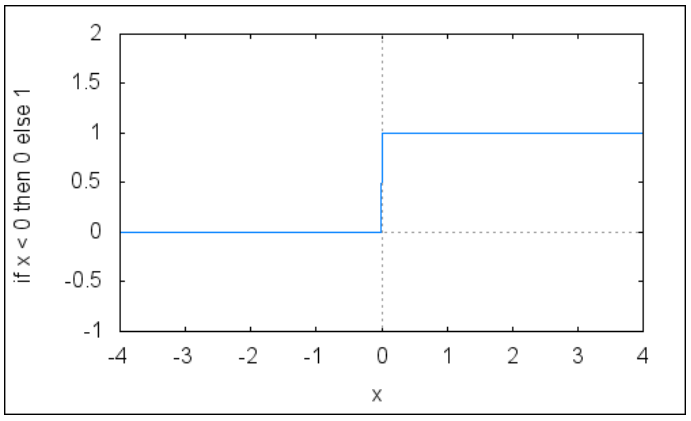
10.3 Test

Test : si, alors, sinon (if, then, else)

```
(%i14) f(x):=if x < 0 then 0 else 1;
```

(%o14) $\mathrm{f}\,(x) := \mathrm{if} x < 0 \mathrm{then} 0 \mathrm{else} 1$

```
(%i15) wxplot2d(f(x),[x, -4,4], [y, -1, 2]);
```



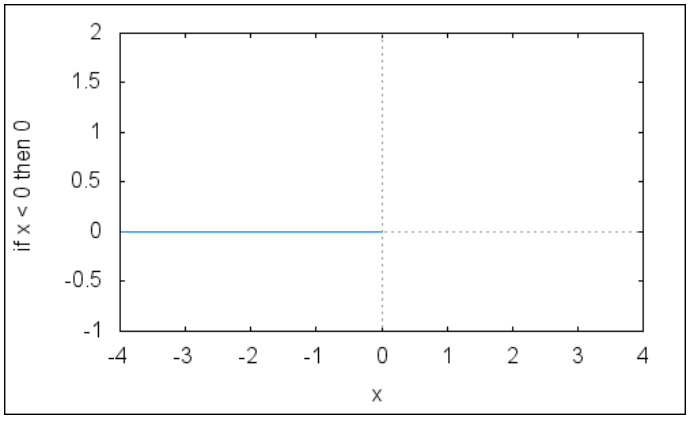
(%t15)

(%o15)

Sans le else :

```
(%i16) f(x):=if x < 0 then 0;
       wxplot2d(f(x),[x, -4,4], [y, -1, 2]);
```

(%o16) $\mathrm{f}\,(x) := \mathrm{if} x < 0 \mathrm{then} 0 plot2d : expressionevaluatestononnumericvaluesomewhereinplottingrange.$



(%t17)

(%o17)

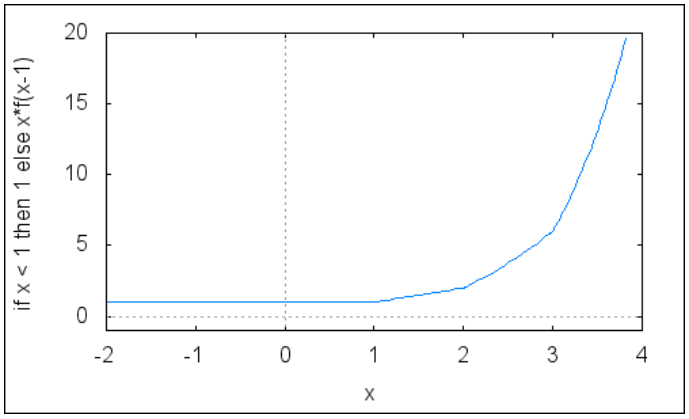
Récursivité définition de la factorielle (pour x entier)

```
(%i18) f(x):= if x < 1 then 1 else x*f(x-1);
```

(%o18) $\mathrm{f}\,(x) := \mathrm{if} x < 1 \mathrm{then} 1 \mathrm{else} x\,\mathrm{f}\,(x-1)$

```
(%i19) wxplot2d(f(x),[x, -2,4], [y, -1, 20]);
```

$plot2d : somevalueswereclipped.$



(%t19)

(%o19)

10.4 Logique (utile pour les tests)

Utilisation de vrai et faux

```
(%i20) true and flase;
```

(%o20) *flase*

```
(%i21) true or flase;
```

(%o21) *true*

Avec des parenthèses : associativité

```
(%i22) (true and flase) or true;
```

(%o22) *true*

Avec des variables

```
(%i23) x:true$
       y:false$
       x and y;
       x or y;
       kill(x, y);
```

(%o25) *false*

(%o26) *true*

(%o27) *done*

10.5 Structure d'une fonction/procédure

Définition d'une fonction (une valeur est retournée par un return)

```
nom(paramètres d'entrées):=block([paramètres locaux],
        block d'instructions 1,
        /* Commentaires */
        block d'instructions 2,
        return(valeur)
    );
```

Exemple : définition de la factorielle (pour x entier).

```
(%i28) g(x) := block([temp],
           temp:1,
           while x > 1 do(
               temp:x*temp,
               x:x-1),
           return(temp))$

(%i29) g(5);
```

(%o29) 120

Définition d'une procédure (la dernière valeur est retournée)

```
nom(paramètres d'entrées):=block([paramètres locaux],
        block d'instructions 1,
        /* Commentaires */
        block d'instructions 2
    );
```

Exemple : définition de la factorielle (pour x entier).

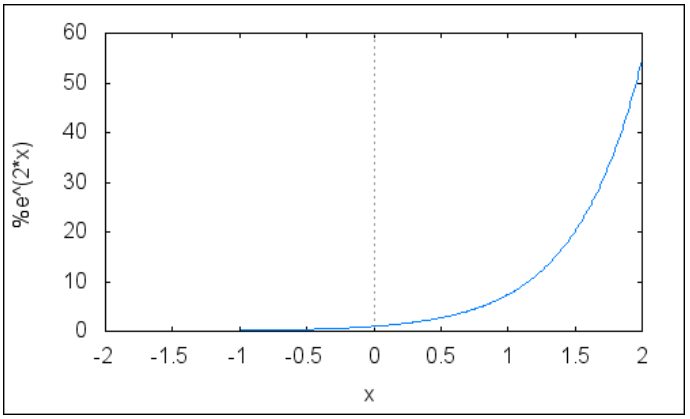
```
(%i30) g(x) := block([temp],
           temp:1,
           while x > 1 do(
               temp:x*temp,
               x:x-1),
           temp)$

(%i31) g(5);
```

(%o31) 120

Un procédure peut être utilisée si l'on souhaite par exemple afficher une courbe et que l'on a rien à retourner.

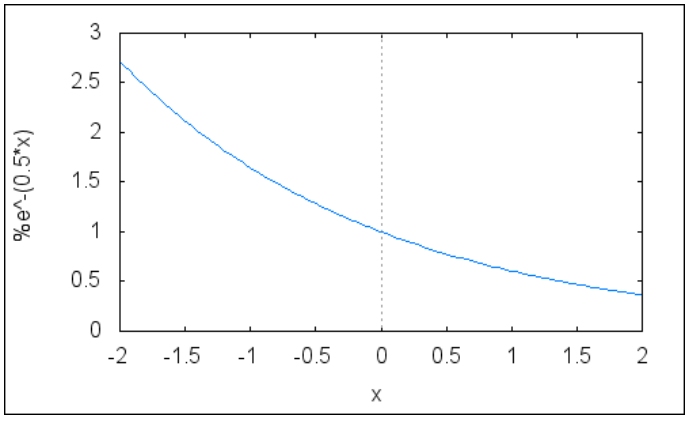
```
(%i32) courbeExponentielle(n) := block([],
           wxplot2d(exp(n*x), [x, -2, 2])
       )$

(%i33) courbeExponentielle(2);
```

(%t33)



(%o33)

```
(%i34) courbeExponentielle(-0.5);
```



(%t34)

(%o34)

10.6 Affichage de chaînes de caractères.

```
(%i35) test(x,y):=block([],
           if x > y then print(x, "est plus grand que ", y)
                    else print(x, "n'est pas plus grand que", y),
           return("Commande exécutée"))$
```

```
(%i36) test(4,3);
```

$4 est plus grand que 3$

(%o36) $Commande exécutée$

Il y a aussi la fonction display qui permet d'afficher le contenu d'une variable.

```
(%i37) x:4;
       display(x);
```

(%o37) $4x = 4$

(%o38) *done*

10.7 Utilisations d'hypothèses

Pour oublier les hypothèses ou les affectations de y :

```
(%i39) kill(y);
```

(%o39) *done*

```
(%i40) assume(y > 0);
```

(%o40) $[y > 0]$

```
(%i41) test(y^2+1,-y);
```

$y^2 + 1 est plus grand que - y$

(%o41) *Commandeexcute*

Pour oublier l'hypothèse :

```
(%i42) forget(y>0);
```

(%o42) $[y > 0]$

```
(%i43) test(y^2+1,-y);
```

(%o43) *Commandeexcute*

```
(%i44) assume(y < 0);
       test(y^2+1,-y);
```

(%o44) $[y < 0]$

(%o45) *Commandeexcute*

```
(%i46) facts();
```

(%o46) $[0 > y]$

```
(%i47) forget(all);
```

(%o47) $[true]$

```
(%i48) facts();
```

(%o48) $[0 > y]$

10.8 Types de variables

Si n est un entier, on peut écrire :

```
(%i49) declare(n, integer);
       assume(n>0);
```

(%o49) *done*

(%o50) $[redundant]$

Cela peut changer des calculs :

```
(%i51) cos(n*%pi);
```

(%o51) 1

Si on oublie tout :

```
(%i52) kill(all);
```

(%o0) *done*

```
(%i1)  cos(n*%pi);
```

(%o1) $\cos(\pi\, n)$

10.9 Commentaires

```
(%i2)  /* Ceci est un commentaire,
          la somme de deux entiers ! */
       1+2;
(%o2)  3
```

Chapitre 11

Les développements limités

11.1 Introduction

Dans ce chapite, on va étudier la fonction f définie ci-dessous au voisinage de 0 :

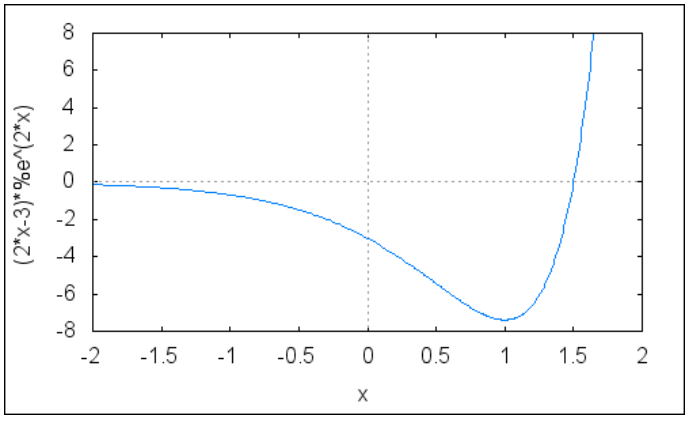
```
(%i1)  define(f(x),(2*x-3)*exp(2*x));
```

(%o1) $\mathrm{f}\,(x) := (2\,x - 3)\;e^{2\,x}$

```
(%i2)  wxplot2d([f(x)], [x,-2,2], [y, -8,8])$
```

plot2d : *somevalueswereclipped.*

(%t2)



11.2 Tangente à une courbe

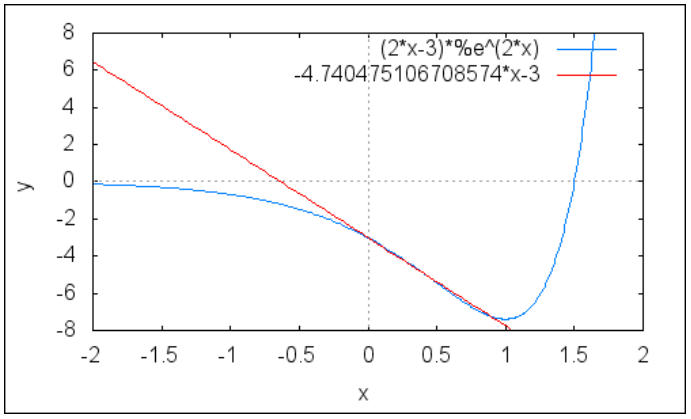
On considère le point $A(0;-3)$ et $M(x;f(x))$ les points de la courbe représentative de la fonction f, respectivement aux points d'abscisse 0 et x. La fonction ci-dessous permet de définie la fonction affine par dont la courbe est la droite (M_0M_1), où $M_0(x_0;y_0)$ et $M_1(x_1;y_1)$.

```
(%i3)  droiteAM(x0, y0, x1, y1):=block([],
           (y1-y0)/(x1-x0)*x+y0
       )$
```

```
(%i4)  wxplot2d([f(x), droiteAM(0,-3,0.4, f(0.4))], [x,-2,2], [y, -8,8])$
```

plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t4)

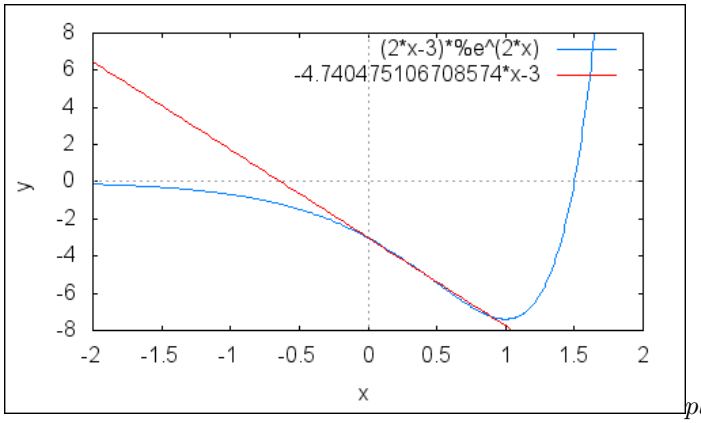


Si le point M se rapproche du point A alors la droite (AM) se rappproche de la tangente à la courbe représentative de f au point A.

```
(%i5)  for i from 1 thru 4 do wxplot2d([f(x), droiteAM(0,-3,0.4^(i), f(0.4^(i)))], [x,-2,2], [y, -8,8
```

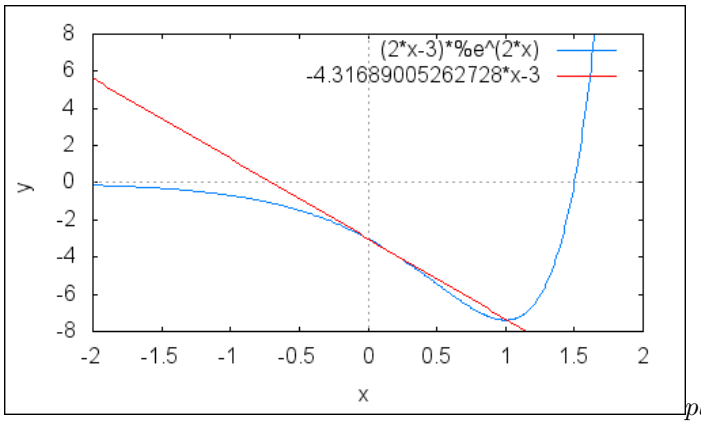
plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t5)

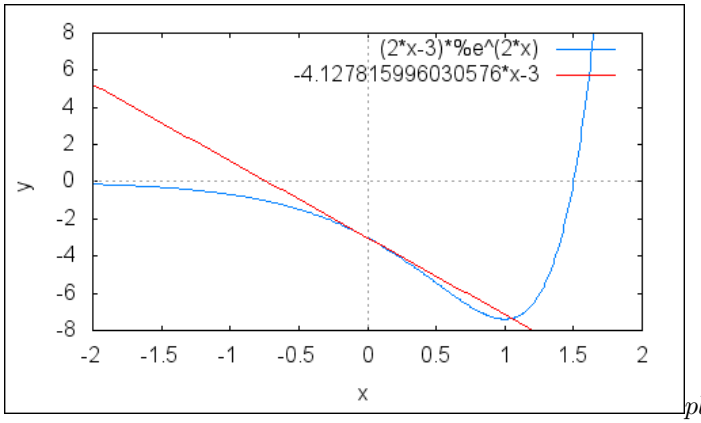


plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

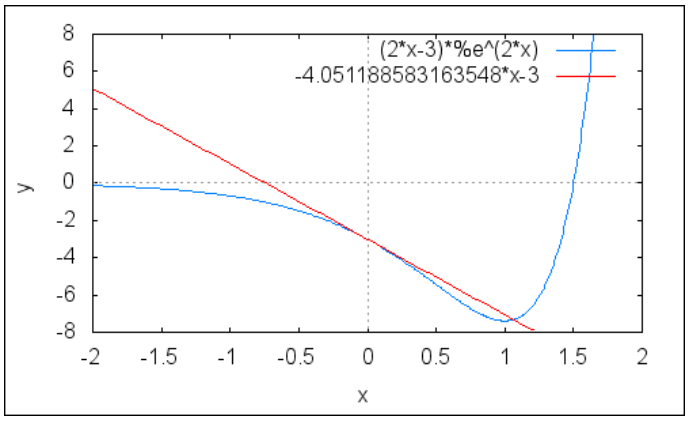
(%t6)



plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t7) 

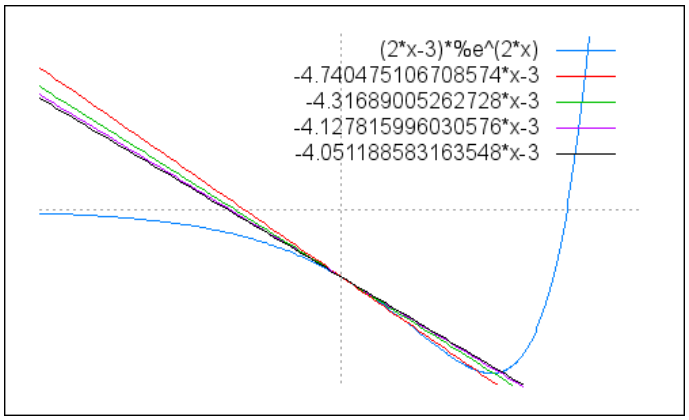
plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t8) 

Il est aussi possible d'afficher toutes les droites sur le même graphique en utilisant les listes :

```
(%i9)  L:[f(x)]$
       for i from 1 thru 4 do L:append(L,[droiteAM(0,-3,0.4^(i), f(0.4^(i)))])$
       wxplot2d(L, [x,-2,2], [y, -8,8], [box, false], [plot_format, xmaxima])$
```

plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t11) 

Que peut-on remarquer ?

11.3 Introduction aux développements limités

Dans ce paragraphe, on souhaite réaliser le même type d'approximation, avec des fonctions polynômes du second degré au lieu des fonctions affines. Pour réaliser cela, on charge en mémoire la bibliothèque interpol.

```
(%i12) load(interpol)$
```

La variable p contient la liste des points par lesquels la courbe du polynôme P du second degré va passer.

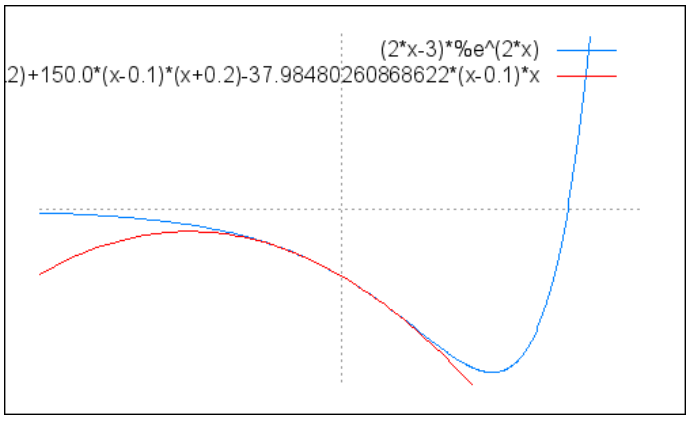
```
(%i13) p:[[0,-3],[0.1,f(0.1)],[-0.2,f(-0.2)]]$
       define(P(x),lagrange(p))$
```

On trace les deux courbes :

```
(%i15) wxplot2d([f(x),P(x)], [x,-2,2], [y, -8,8], [box, false], [plot_format, xmaxima])$
```

plot2d : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t15)

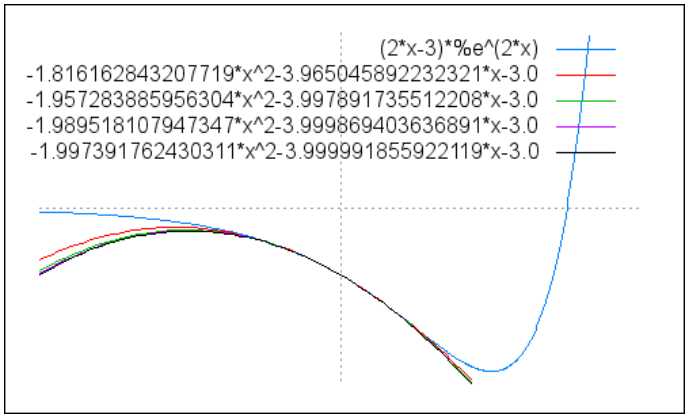


De la même façon que pour les polynômes du second degré, on trace différentes approximations :

```
(%i16) L:[f(x)]$
       for i from 1 thru 4 do L:append(L,[expand(lagrange([[0,-3],[0.5^i,f(0.5^i)],[-0.5^i,f(-0.5^i)]
       L;
       wxplot2d(L, [x,-2,2], [y, -8,8], [box, false], [plot_format, xmaxima])$
```

(%o18) $[(2\,x-3)\,e^{2\,x}, -1.816162843207719\,x^2-3.965045892232321\,x-3.0, -1.957283885956304\,x^2-3.997891735512208\,x-3.0, -1.989518107947347\,x^2-3.999869403636891\,x-3.0, -1.997391762430311\,x^2-3.999991855922119\,x-3.0]$*plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.plot2d* : *somevalueswereclipped.*

(%t19)



11.4 Développements limités des fonctions usuelles

On note N l'ordre des développements que l'on veut étudier.

```
(%i20) N:8;
```

(%o20) 8

La fonction exponentielle :

```
(%i21) taylor(exp(x), x, 0, N);
```

(%o21)/T/ $x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + \ldots$

La fonction inverse :

```
(%i22) taylor(1/(1+x), x, 0, N);
```

(%o22)/T/ $x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 + \ldots$

La fonction logarithme népérien :

```
(%i23) taylor(log(1+x), x, 0, N);
```

(%o23)/T/ $\frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \frac{x^6}{6} + \frac{x^7}{7} - \frac{x^8}{8} + \ldots$

La fonction sinus :

```
(%i24) taylor(sin(x), x, 0, N);
```

(%o24)/T/ $\frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots$

La fonction cosinus :

```
(%i25) taylor(cos(x), x, 0, N);
```

(%o25)/T/ $\frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40320} + \ldots$

La fonction puissance :

```
(%i26) taylor((1+x)^alpha, x, 0, N);
```

$$(\%o26)/T/\alpha\, x+\frac{\left(\alpha^2-\alpha\right)\, x^2}{2}+\frac{\left(\alpha^3-3\,\alpha^2+2\,\alpha\right)\, x^3}{6}+\frac{\left(\alpha^4-6\,\alpha^3+11\,\alpha^2-6\,\alpha\right)\, x^4}{24}+\frac{\left(\alpha^5-10\,\alpha^4+35\,\alpha^3-50\,\alpha^2+24\,\alpha\right)\, x}{120}$$
$$\frac{\left(\alpha^6-15\,\alpha^5+85\,\alpha^4-225\,\alpha^3+274\,\alpha^2-120\,\alpha\right)\, x^6}{720}+\frac{\left(\alpha^7-21\,\alpha^6+175\,\alpha^5-735\,\alpha^4+1624\,\alpha^3-1764\,\alpha^2+720\,\alpha\right)\, x^7}{5040}+$$
$$\frac{\left(\alpha^8-28\,\alpha^7+322\,\alpha^6-1960\,\alpha^5+6769\,\alpha^4-13132\,\alpha^3+13068\,\alpha^2-5040\,\alpha\right)\, x^8}{40320}+\ldots$$

Chapitre 12

Fonctions et Modélisation du signal

12.1 La fonction tangente

12.1.1 Définition

```
(%i1)  define(f(x),tan(x));
```

(%o1) $\mathrm{f}(x) := \tan(x)$

12.1.2 Tableau de valeurs de la fonction tangente

```
(%i2)  valeursdex:sort(append(makelist(k*%pi/6,k,0, 12),[%pi/4, 3*%pi/4, 5*%pi/4, 7*%pi/4]))$
       valeursdex:delete(%pi/2,valeursdex)$
       valeursdex:delete(3*%pi/2,valeursdex);
       valeursdetanx:map(f,valeursdex);
```

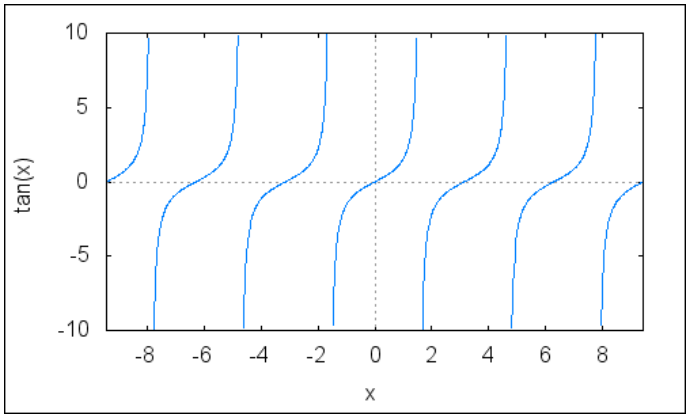
(%o4) $[0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{2\,\pi}{3}, \frac{3\,\pi}{4}, \frac{5\,\pi}{6}, \pi, \frac{7\,\pi}{6}, \frac{5\,\pi}{4}, \frac{4\,\pi}{3}, \frac{5\,\pi}{3}, \frac{7\,\pi}{4}, \frac{11\,\pi}{6}, 2\,\pi]$

(%o5) $[0, \frac{1}{\sqrt{3}}, 1, \sqrt{3}, -\sqrt{3}, -1, -\frac{1}{\sqrt{3}}, 0, \frac{1}{\sqrt{3}}, 1, \sqrt{3}, -\sqrt{3}, -1, -\frac{1}{\sqrt{3}}, 0]$

12.1.3 Courbe

```
(%i6)  wxplot2d(f(x), [x, -3*%pi,3*%pi], [y, -10,10]);
```

plot2d : somevalueswereclipped.



(%t6)

(%o6)

12.1.4 Dérivées

```
(%i7)  diff(f(x),x);
```

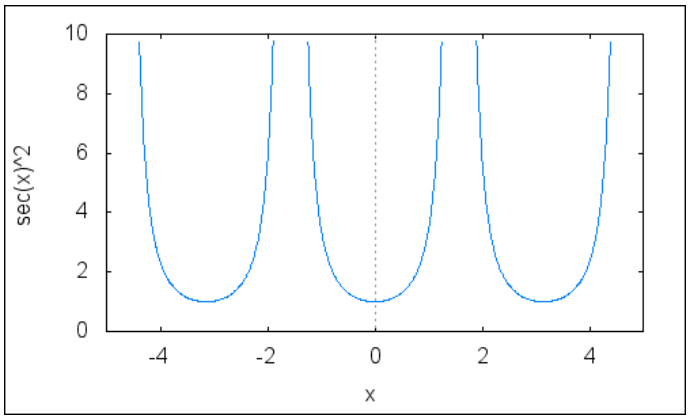
(%o7) $\sec(x)^2$

```
(%i8)  trigsimp(diff(f(x),x));
```

(%o8) $\frac{1}{\cos(x)^2}$

```
(%i9)  wxplot2d(sec(x)^2, [x,-5,5],[y, 0,10]);
```

plot2d : somevalueswereclipped.



(%t9)

(%o9)

```
(%i10) trigsimp(diff(f(u(x)),x));
```

(%o10) $\frac{\frac{d}{dx}\,\mathrm{u}(x)}{\cos(\mathrm{u}(x))^2}$

12.2 La fonction arctangente

12.2.1 Définition

```
(%i11) define(f(x),atan(x));
```

(%o11) $\mathrm{f}\left(x\right) := \mathrm{atan}\left(x\right)$

12.2.2 Tableau de valeurs

```
(%i12) valeursdex:[-sqrt(3),-1,-1/sqrt(3),0,1/sqrt(3),1,sqrt(3)];
       valeursdeatanx:map(f,valeursdex);
```

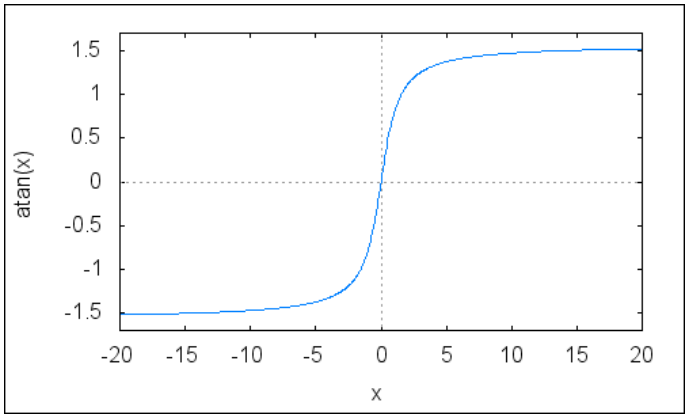
(%o12) $[-\sqrt{3}, -1, -\frac{1}{\sqrt{3}}, 0, \frac{1}{\sqrt{3}}, 1, \sqrt{3}]$

(%o13) $[-\frac{\pi}{3}, -\frac{\pi}{4}, -\frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}]$

12.2.3 Courbe

```
(%i14) wxplot2d(f(x), [x, -20,20], [y, -1.7,1.7]);
```

(%t14)



(%o14)

12.2.4 Dérivées

```
(%i15) diff(f(x),x);
```

(%o15) $\frac{1}{x^2+1}$

```
(%i16) diff(f(u(x)),x);
```

(%o16) $\frac{\frac{d}{dx}\,\mathrm{u}\left(x\right)}{\mathrm{u}\left(x\right)^2+1}$

12.3 Fonctions paires et impaires

Vérifier qu'une fonction est paire

```
(%i17) f(x):=x^2;
       if (f(x)=f(-x)) then "f est paire" else "f n'est pas paire";
       g(x):=x^3;
       if (g(x)=g(-x)) then "g est paire" else "g n'est pas paire";
```

(%o17) $\mathrm{f}(x) := x^2$

(%o18) $festpaire$

(%o19) $\mathrm{g}(x) := x^3$

(%o20) $gn'estpaspaire$

Vérifier qu'une fonction est impaire

```
(%i21) f(x):=x^2;
       if (f(x)=-f(-x)) then "f est impaire" else "f n'est pas impaire";
       g(x):=sin(x);
       if (g(x)=-g(-x)) then "g est impaire" else "g n'est pas impaire";
```

(%o21) $\mathrm{f}(x) := x^2$

(%o22) $fn'estpasimpaire$

(%o23) $\mathrm{g}(x) := \sin(x)$

(%o24) $gestimpaire$

Les deux à la fois

```
(%i25) parite(f):=block([],
       if (f(x)=f(-x)) then "fonction paire" else (if (f(x)=-f(-x)) then "fonction impaire" else "for
       )$

(%i26) f(x):=x^2;
       parite(f);
```

(%o26) $\mathrm{f}(x) := x^2$

(%o27) $fonctionpaire$

```
(%i28) f(x):=x^3;
       parite(f);
```

(%o28) $\mathrm{f}(x) := x^3$

(%o29) $fonctionimpaire$

```
(%i30) f(x):=sin(x);
       parite(f);
```

(%o30) $\mathrm{f}(x) := \sin(x)$

(%o31) $fonctionimpaire$

```
(%i32) f(x):=cos(x)+sin(x);
       parite(f);
```

(%o32) $\mathrm{f}(x) := \cos(x) + \sin(x)$

(%o33) *fonctionnipaireniimpaire*

Définition d'une fonction paire

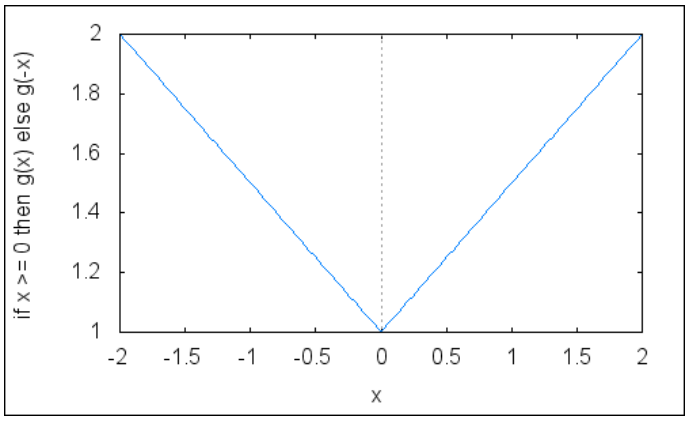
```
(%i34) a:2;
       g(x):=1/2*x+1;
       define(f(x),if x>=0 then g(x) else g(-x));
       wxplot2d([f(x)], [x,-a,a])$
```

(%o34) 2

(%o35) $\mathrm{g}\left(x\right) := \frac{1}{2}\,x + 1$

(%o36) $\mathrm{f}\left(x\right) := \mathrm{if}\, x >= 0 \mathrm{then}\, \left(x\right) \mathrm{else}\, \left(-x\right)$

(%t37)



Définition d'une fonction impaire

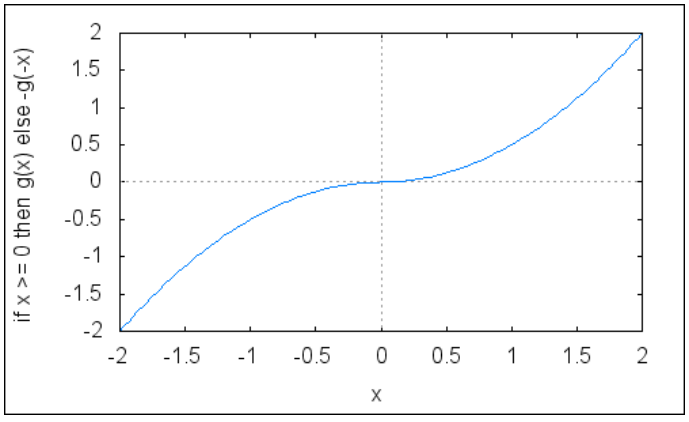
```
(%i38) a:2;
       g(x):=1/2*x^2;
       define(f(x),if x>=0 then g(x) else -g(-x));
       wxplot2d([f(x)], [x,-a,a])$
```

(%o38) 2

(%o39) $\mathrm{g}\left(x\right) := \frac{1}{2}\,x^2$

(%o40) $\mathrm{f}\left(x\right) := \mathrm{if}\, x >= 0 \mathrm{then}\, \left(x\right) \mathrm{else} - \mathrm{g}\left(-x\right)$

(%t41)



12.4 Fonctions périodiques

12.4.1 Un exemple

```
(%i42) T:2*%pi;
```

(%o42) $2\,\pi$

```
(%i43) f(x):=sin(x);
```

(%o43) $\mathrm{f}\left(x\right) := \sin\left(x\right)$

```
(%i44) if(f(x)=f(x+T)) then sconcat("f est périodique de période ",T) else sconcat("f n'est pas pério
```

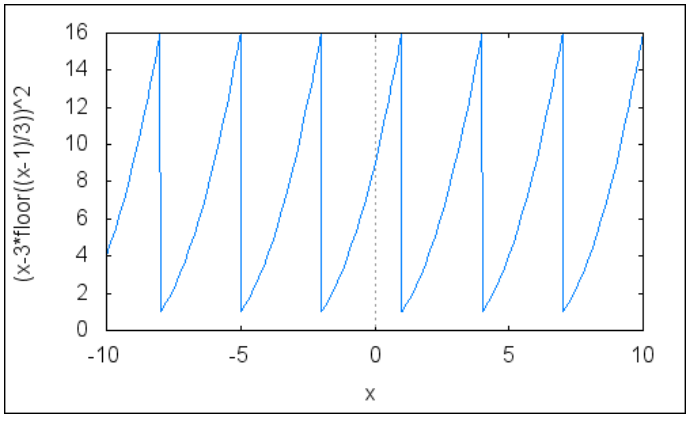
(%o44) $festpriodiquedepriode2 * \%pi$

12.4.2 Définition d'une fonction périodique

La fonction est définie sur l'intervalle [a,b] puis prolongée sur R par périodicité

```
(%i45) a:1$
       b:4$
       g(x):=x^2$
       f(x):=g(x-(b-a)*floor((x-a)/(b-a)));
       wxplot2d([f(x)], [x, -10, 10]);
```

(%o48) $\mathrm{f}\left(x\right) := \mathrm{g}\left(x - \left(b - a\right)\ \mathrm{floor}\left(\frac{x - a}{b - a}\right)\right)$



(%t49)

(%o49)

12.5 Fonctions rationnelles

On considère une fonction rationnelle dont le dénominateur est factorisable en produit de facteurs linéaires et quadratiques (dont on sait calculer la valeur exacte des racines).

```
(%i50) f(x):=(x^7+3*x^2-5*x+1)/(x^5-11*x^4+45*x^3-85*x^2+74*x-24);
```

(%o50) $\mathrm{f}\left(x\right) := \frac{x^7 + 3\,x^2 + \left(-5\right)\ x + 1}{x^5 - 11\,x^4 + 45\,x^3 + \left(-85\right)\ x^2 + 74\,x - 24}$

Décomposition en éléments simples

```
(%i51) partfrac(f(x),x);
```

$$(\%o51)\ x^2 + 11\,x - \frac{4}{3\,(x-1)} + \frac{131}{2\,(x-2)} - \frac{550}{x-3} + \frac{5471}{6\,(x-4)} + 76$$

Chapitre 13

Approximations globales

13.1 Première méthode : à l'aide d'une approximation locale

On utilise la notion de tangente et on essaie de la généraliser à des polynômes du second degré, puis à des polynômes de degré supérieur. On considère f une fonction définie et indéfiniment dérivable sur R et a un nombre réel quelconque.

```
(%i1)  define(f(x),exp(x));
       a:2;
```

(%o1) $\mathrm{f}\left(x\right) := e^{x}$

(%o2) 2

On note df sa fonction dérivées

```
(%i3)  define(df(x), diff(f(x),x));
```

(%o3) $\mathrm{df}\left(x\right) := e^{x}$

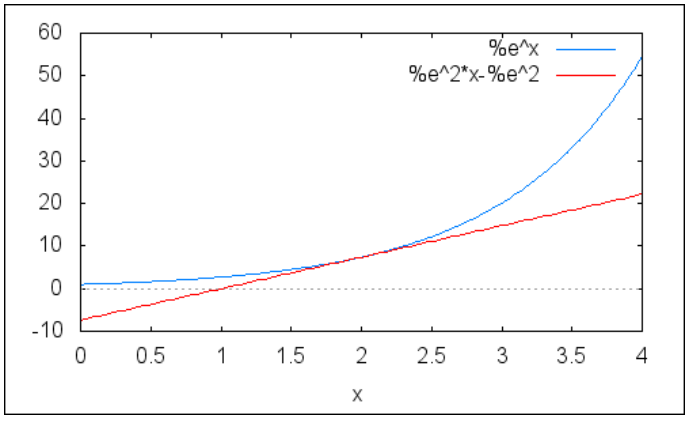
On connaît l'équation de la tangente à la courbe représentative de la fonction f au point a.

```
(%i4)  T(x):=expand(df(a)*(x-a)+f(a));
```

(%o4) $\mathrm{T}\left(x\right) := \mathrm{expand}\left(\mathrm{df}\left(a\right)\ \left(x-a\right)+\mathrm{f}\left(a\right)\right)$

```
(%i5)  wxplot2d([f(x), T(x)], [x, 0, 4]);
```

(%t5)



(%o5)

On cherche à approcher la fonction f par un polynôme de degré 2 au voisinage de f. Notons α le coefficient de x^2.

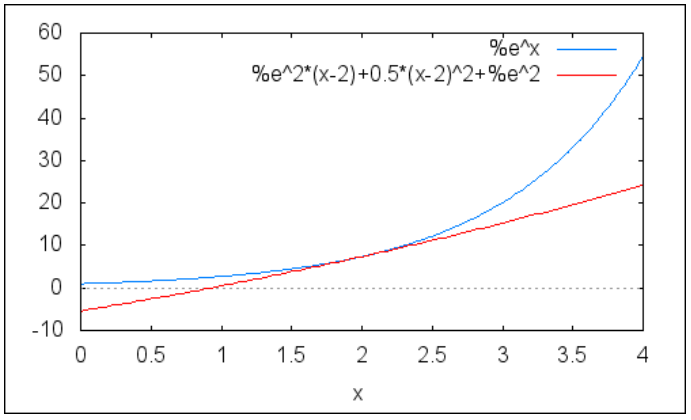
```
(%i6)  %alpha:0.5;
       T2(alpha, x):=alpha *(x-a)^2+df(a)*(x-a)+f(a);
```

(%o6) 0.5

(%o7) $\mathrm{T2}\left(\alpha, x\right) := \alpha\left(x-a\right)^2 + \mathrm{df}\left(a\right)\ \left(x-a\right) + \mathrm{f}\left(a\right)$

```
(%i8)  wxplot2d([f(x), T2(0.5,x)], [x, 0, 4]);
```

(%t8)



(%o8)

Quel est le meilleur choix de α ? Pour répondre à cette question, on renvoi le lecteur au chapitre sur les développements limités, cas où l'approximation est locale.

13.2 Interpolation de Lagrange

On considère f une fonction définie et indéfiniment dérivable sur R et a un nombre réel quelconque. On cherche une fonction polynôme g dont la courbe représentative va passer par des points définis à l'avance et qui permettent d'approximer la fonction f. C'est l'interpolation de Lagrange.

```
(%i9)  load(interpol)$
```

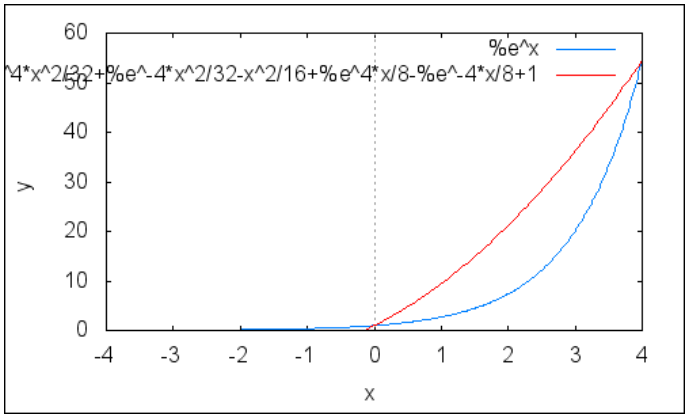
Avec 3 points :

```
(%i10) p:[[-4,f(-4)],[0,f(0)],[4,f(4)]]$
       define(g(x),expand(lagrange(p)));
```

(%o11) $\mathrm{g}\left(x\right) := \dfrac{e^4\,x^2}{32} + \dfrac{e^{-4}\,x^2}{32} - \dfrac{x^2}{16} + \dfrac{e^4\,x}{8} - \dfrac{e^{-4}\,x}{8} + 1$

```
(%i12) wxplot2d([f(x), g(x)], [x, -4, 4],  [y, 0, 60]);
```

plot2d : somevalueswereclipped.



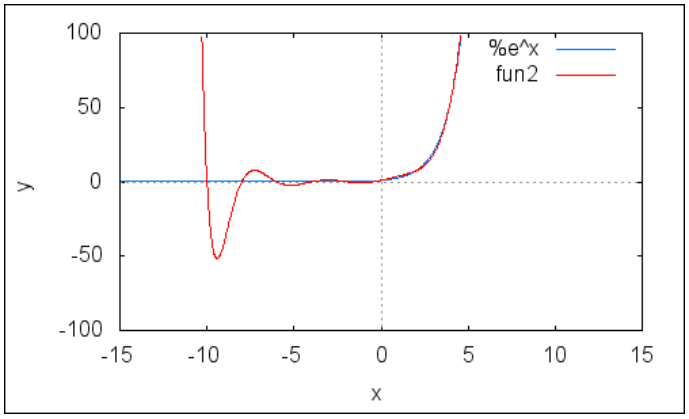
(%t12)

(%o12)

Avec autant de points que souhaité :

```
(%i13) N:10$
       a:-10$
       b:10$
       pas:(b-a)/N$
       p:[]$
       for i from a step pas thru b do p:append(p, [[i, f(i)]])$
       define(g(x),expand(lagrange(p)))$
       wxplot2d([f(x), g(x)], [x, -15, 15],  [y, -100, 100]);
```

plot2d : somevalueswereclipped.plot2d : somevalueswereclipped.



(%t20)

(%o20)

La qualité de l'approximation est très relative...

Chapitre 14

Équations différentielles

14.1 Équations différentielles du premier ordre

On définit une équation différentielle et son équation homogène :

```
(%i1)  equadiff:'diff(y,x)+y=exp(x);
       equadiff0:lhs(equadiff)=0;
```

(%o1) $\frac{d}{d\,x}\,y + y = e^{x}$

(%o2) $\frac{d}{d\,x}\,y + y = 0$

Menu/Équations/Résoudre une équation différentielle...

```
(%i3)  ode2(equadiff, y, x);
       ode2(equadiff0, y, x);
```

(%o3) $y = e^{-x}\left(\frac{e^{2\,x}}{2} + \%c\right)$

(%o4) $y = \%c\,e^{-x}$

où la fonction inconnue est notée y, sa dérivée 'diff(y,x) et %c désigne un réel quelconque. On peut développer le résultat précédent en utilisant % et expand (% permet de récupérer le dernier résultat calculé) :

```
(%i5)  expand(%);
```

(%o5) $y = \%c\,e^{-x}$

Si par exemple on a la condition initiale y(0)=1 : Menu/Équations/Condition initiele (1)...

```
(%i6)  ic1(%, x=0, y=1);
```

(%o6) $y = e^{-x}$

On peut développer :

```
(%i7)  expand(%);
```

(%o7) $y = e^{-x}$

On peut vérifier que l'on obtient bien une solution de l'équation différentielle. La commande suivante remplace y par son expression dans l'équation différentielle, diff permet de calculer les dérivées.

```
(%i8)  ev(equadiff, %o6, diff);
```

(%o8) $0 = e^x$

On a l'égalité donc c'est bien une solution de l'équation différentielle.

14.2 Équations différentielles du second ordre

Menu/Équations/Résoudre une équation différentielle...

```
(%i9)  ode2('diff(y,x,2)+4*'diff(y,x)+4*y=1-exp(-x), y, x);
```

(%o9) $y = \frac{e^{-x}\,(e^x-4)}{4} + (\%k2\,x + \%k1)\,e^{-2\,x}$

où la fonction inconnue est notée y, sa dérivée 'diff(y,x), sa dérivée seconde 'diff(y,x,2) et %k1 et %k2 désignent deux réels quelconques. On peut développer

```
(%i10) expand(%o9);
```

(%o10) $y = -e^{-x} + \%k2\,x\,e^{-2\,x} + \%k1\,e^{-2\,x} + \frac{1}{4}$

Si on a les conditions initiales y(0)=1 et y'(0)=2 : Menu/Équations/Condition initiele (2)...

```
(%i11) ic2(%o9, x=0, y=1, 'diff(y,x)=2);
```

(%o11) $y = \frac{e^{-x}\,(e^x-4)}{4} + \left(\frac{9\,x}{2} + \frac{7}{4}\right)\,e^{-2\,x}$

Si on a les conditions initiales y(0)=1 et y(1)=3 : Menu/Équations/Problème aux limites...

```
(%i12) bc2(%o9,x = 0,y = 1,x = 1,y = 3);
```

(%o12) $y = \frac{e^{-x}\,(e^x-4)}{4} + \left(\frac{(11\,e^2+4\,e-7)\,x}{4} + \frac{7}{4}\right)\,e^{-2\,x}$

14.3 Exemple : masse - ressort - frottement visqueux

```
(%i13) equation:m*'diff(y,x,2)+c*'diff(y,x)+r*y=1;
```

(%o13) $m\left(\frac{d^2}{d\,x^2}\,y\right) + c\left(\frac{d}{d\,x}\,y\right) + r\,y = 1$

Pour delta positif où m est la masse, c est le coefficient de frottement, r la raideur, et le second membre de l'équation différentielle est donné par une force externe appliquée à la masse

```
(%i14) forget(facts());
       assume(c**2-4*m*r>0);
       ode2(equation, y, x);
```

(%o14) [[]]