IMPLÉMENTATION DE LA MÉTHODE DE FOURIER POUR LE CALCUL DES POIDS DE MAYER

Amel Kaouche, Université de Moncton Campus d'Edmundston, Canada

Gilbert Labelle, LaCIM, Université du Québec à Montréal, Canada

Nous présontons ici des procédures en Maple qui implémentent la méthode de Fourier pour le calcul du poids de Mayer. La procédure principale wMayer prend en entrée les deux paramètres a et ca où a est une arborescence couvrante d'un graphe connexe c donnée sous la forme d'une liste

$$a = [\alpha(1), \alpha(2), \ldots, \alpha(k)], \qquad \alpha(i) < i, \tag{1}$$

et son complémentaire ca donné par

$$ca = c \backslash a = [\{i,j\} | \{i,j\} \text{ arête de } c \text{ non présente dans } a\ ] = ca \text{ (notation)}. \tag{2}$$

0.1 Calcul de la différence divisée pour des points distincts

Dans le programme qui suit, la procédure *dd* calcule la différence divisée de expr pour la liste de points distincts lst par rapport à la variable var.

```
dd := proc(expr, lst, var) local i, p, n;
 n := nops(lst);
for i from 1 to n
do
p[i] := mul(lst[i]-lst[j],j = 1..i-1)*mul(lst[i]-lst[j],j = i+1..n)
od;
add(subs(var=lst[i],expr)/p[i], i=1..n);
end:
```

0.2 Calcul de la différence divisée pour des points répétés

Dans le programme qui suit, la procédure *DD* calcule la différence divisée de expr pour le multi-ensemble mset par rapport à la variable var dans le cas confluent.

```
DD := proc(expr, mset, var) local i, p, lstv, v, lstm, n, aux;
n :=nops(mset);
lstv := [seq(op([i,1],mset),i=1..n)];
```

```
lstm := [seq(op([i,2],mset),i=1..n)];
aux := dd(expr,lstv,var);
for i from 1 to n do
if lstm[i]>1 then
aux := diff(aux,lstv[i]$(lstm[i]-1))/(lstm[i]-1)! fi; od;
RETURN(aux);
end:
```

0.3 Réduction d'un terme en un autre qui utilise seulement $\exp(i\nu t)$ où $\nu \geq 0$

Dans le programme qui suit, la procédure *clean* implémente la remarque sur l'accélération. Elle prend the_term:= $c_\nu(t)e^{i\nu t}$ dans T et le transforme en $2c_\nu(t)e^{i\nu t}$, si $\nu > 0$, en $c_0(t)e^{i0t}$, si $\nu = 0$ ou en 0, si $\nu < 0$ (var:=t).

```
clean := proc(the_term, var);
if limit(subs(var=I*var, the_term), var=infinity) = 0 then
RETURN(2*the_term)
elif limit(subs(var=-I*var, the_term), var=infinity) = 0 then
RETURN(0)
else RETURN(the_term) fi;
end:
```

0.4 Réduction de l'intégrand en utilisant seulement $\exp(i\nu t)$ où $\nu \geq 0$

Dans le programme qui suit, la procédure *U_of_T* utilise la fonction clean en transformant T en U où (var:=t).

```
U_of_T := proc(z, var) local aux;
aux := combine(expand(convert(z, exp)));
RETURN( map(clean,aux,var) )
end:
```

0.5 Calcul de l'intégrale de T/P

Dans le programme qui suit, la procédure *compute* intégre TP (:=T/P) par rapport à la variable (var:=t).

```
compute := proc(TP,var) local sol, T, U_T, P, lc, tau, sol_tau,z;
T := numer(TP); P := denom(TP);
T := combine(T,trig); U_T := U_of_T(T,var);
lc := lcoeff(P,var);
sol := convert([solve(P,var)], multiset);
sol_tau := [seq([tau[i-1],sol[i][2]], i=1..nops(sol))];
z := DD(U_T,sol_tau,var);
```

```
z := subs([seq(tau[i-1]=sol[i][1], i=1..nops(sol))],z);
RETURN(combine((I/2)*normal(z/lc), trig));
end:
```

0.6 Calcul de l'ensemble des descendants d'un sommet j dans l'arborescence a

Dans le programme qui suit, la procédure *desc* calcule l'ensemble des descendants du sommet j dans l'arborescence a.

```
desc := proc(j, a) local aux, ind;
aux := {j}; ind := j;
while ind <> 0
 do aux := aux union {a[ind]};
    ind := a[ind]
od; RETURN(aux)
end:
```

0.7 Calcul de l'ensemble des sommets de la sous-arborescence induite a_ν

Dans le programme qui suit, la procédure *vert* calcule l'ensemble des sommets de la sous-arborescence a_ν en prenant en entrée a et ν.

```
vert := proc(a,nu) local aux, j;
aux := {};
for j from 0 to nops(a)
 do
  if evalb(nu in desc(j,a)) then aux := aux union {j} fi;
 od;
RETURN(aux);
end:
```

0.8 Calcul de la différence entre les variables entrantes et sortantes

Dans le programme qui suit, la procédure *sum_diff* calcule $\sum_{a_\nu \prec e} t_e - \sum_{e \prec a_\nu} t_e$ en prenant en entrée l'arborescence a, son complément ca et le sommet ν.

```
sum_diff := proc(a, ca, nu) local e, aux;
aux := 0;
for e in ca
do
  if evalb( min(op(e)) in vert(a, nu) ) then aux := aux+t[e] fi;
  if evalb( max(op(e)) in vert(a, nu) ) then aux := aux-t[e] fi;
```

```
od;
RETURN(aux);
end:
```

0.9 La transformée de Fourier de $\chi(x \in [-1,1])$

Dans le programme qui suit, la procédure s est la fonction

$$x \to \begin{cases} \frac{2\sin x}{x} & x \neq 0, \\ 2 & x = 0, \end{cases}$$

qui est la transformée de Fourier de $\chi\left(|x| \leq 1\right)$.

```
s := x -> if x=0 then 2 else 2*sin(x)/x  fi:
```

0.10 Calcul de l'intégrand pour le poids de Mayer de c

Dans le programme qui suit, la procédure *theTP* calcule l'intégrand qui donne le poids de Mayer $w_M(c)$ où c est donné en entrées sous la forme (a, ca).

```
theTP := proc(a,ca) local p, nu;
p := mul(s(t[e]), e in ca);
for nu from 1 to nops(a) do p := p*s(sum_diff(a,ca,nu)) od;
RETURN(p);
end:
```

0.11 Représentation des sum_diff comme des multi-ensembles

Dans le programme qui suit, la procédure *the_items* représente la famille des "sum_diff" comme multi-ensemble en prenant en entrée l'arborescence a et son complément ca.

```
 the_items := proc(a,ca) local e, nu,item; global t;
item := NULL:
for e in ca do item := item,abs(t[e]) od;
for nu from 1 to nops(a)
do
item := item, abs(sum_diff(a,ca,nu))
od;
item := convert([item], multiset);
RETURN(convert(item, set));
end:
```

0.12 Calcul du poids de Mayer

Dans le programme qui suit, la procédure principale *wMayer* utilise toutes les procédures précédentes et calcule le poids de Mayer d'un graphe c en prenant en entrée l'arborescence a et son complement ca.

```
wMayer := proc(a,ca) local A, items,e, it; global t;
items := the_items(a,ca); A := 1;
for e in ca
do
for it in items
do if has(it,t[e]) then
 A := A*s(op(op(1,it)))^op(2,it); items := items minus {it} fi;
 od;
 A := compute(A, t[e]);
od;
RETURN(simplify(A));
end:
```

0.13 Calcul du poids de Mayer des échelles

Le programme qui suit illustre l'utilisation de la procédure *wMayer* en calculant le poids des échelles Échelle[k], pour $k = 2, 3, \ldots 20$. C'est ainsi que la table 4 a été produite.

```
for k from 2 to 20 do
a := [seq(2*floor((i-1)/2),i=1..2*k),2*k]:
ca := [seq({2*i-1, 2*i+1}, i=1..k)]:
w_ladder[k] := wMayer(a,ca):
tt := time():
print(k,evalf(w_ladder[k]),w_ladder[k]);
time()-tt;
od;
```