Alexis Haraux
alexis.haraux@student.ecp.fr

Fiche récapitulative

1. Calcul numérique et formel

Opérateurs mathématiques	
+	addition
-	soustraction
*	multiplication
** ou ^	exposant
@	composition de fonctions
mod	modulo

Opérateurs de comparaison	
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
=	égal
<>	différent

Opérateurs logiques	
and	opérateur logique ET
or	opérateur logique OU

Fonctions mathématiques	
cos(x)	cosinus
sin(x)	sinus
tan(x)	tangente
cosh(x)	cosinus hyperbolique
sinh(x)	sinus hyperbolique
tanh(x)	tangente hyperbolique
arccos(x)	arccosinus
arcsin(x)	arcsinus
arctan(x)	arctangente
arccosh(x)	arccosinus hyperbolique
arcsinh(x)	arcsinus hyperbolique
arctanh(x)	arctangente hyperbolique
exp(x)	exponentielle
log(x) ou ln(x)	logarithme népérien
log[10](x)	logarithme décimal
log[b](x)	logarithme en base b

Nombres	
abs(x)	valeur absolue
floor(x)	partie entière
ceil(x)	partie entière supérieure
round(x)	arrondi
trunc(x)	troncature
min(a,b)	minimum de a et b
max(a,b)	maximum de a et b
sqrt(x)	racine carré
surd(x,n)	racine nième de x
iquo(a,b)	quotient de la division euclidienne de a par b
irem(a,b)	reste de la division euclidienne de a par b
ifactor(n)	factorisation en facteurs premiers
gcd(a,b)	PGCD
lcm(a,b)	PPCM
isprime(n)	test de primalité
evalf(x)	evalue la valeur approchée de x

Alexis Haraux
alexis.haraux@student.ecp.fr

Complexes	
I	i (racine de -1)
polar(r,angle)	le complexe r*exp(i*angle) (à évaluer)
evalc(z)	forme algébrique (a+ib)
abs(z)	module
arg(z)	argument
Re(z)	partie réelle
Im(z)	partie imaginaire
conjugate(z)	conjugué

Variables	
a:=1;	affectation
a:='a';	relâchement (a n'a plus aucune valeur)
restart;	efface toutes les variables

Calcul avancé		
sum(expr(k),k=a..b);	somme	ex: sum(2*k, k=1..5); -> 2 + 4 + 6 + 8 + 10
product(expr(k),k=a..b);	produit	ex: product(2*k, k=1..5); -> 2 * 4 * 6 * 8 * 10

Polynômes	
expand(P)	développement
factor(P) factor(P,complex)	factorisation factorisation dans **C**
sort(P,x)	tri des termes de P par degré décroissant, par rapport à x
collect(P,x)	tri et regroupement des termes de P par degré décroissant, par rapport à x
coeff(P,x,n)	coefficient du terme de degré n
degree(P)	degré du polynôme
lcoeff(P)	coefficient du terme de plus haut degré
gcd(P,Q)	PGCD de P et Q
lcm(P,Q)	PPCM de P et Q

Alexis Haraux
alexis.haraux@student.ecp.fr

Fonctions	
f:= x -> f(x); ex : f:=x->x^2;	définition de f
readlib(singular); singular(f(x));	ensemble des points singuliers de f
iscont(f(x),x=a..b)	continuité de f sur l'intervalle [a,b]
discont(f(x),x)	points de discontinuités de f
limit(f(x),x=a) limit(f(x),x=a,left) limit(f(x),x=a,right)	limite de f au point a limite à gauche de f au point a limite à droite de f au point a
diff(expression,var) ex : diff(f(x),x) diff(f(x),x,x) diff(f(x),x$n)	dérivation d'une expression par rapport à x dérivée seconde de l'expression dérivée n-ième de l'expression
D(f) (D@@n)(f)	fonction dérivée de f (f' en maths) fonction dérivée n-ième de f
int(f(x),x)	primitive de f
int(f(x),x=a..b)	intégrale de f entre a et b
plot(f(x),x=a..b)	trace la fonction sur l'intervalle [a,b]

Alexis Haraux
alexis.haraux@student.ecp.fr

2. Programmation

Définition d'une procédure :

```
nom_procedure := proc(entree1,entree2,...)
        local variable_locale1, variable_locale2,...;
        global variable_globale1, variable_globale2,...;
        instructions;
end;
```

- Les variables d'entrée (entree1, entree2, ...) sont les arguments de la procédure, ce sont les informations que vous donnez à la procédure pour qu'elle effectue son calcul (c'est l'équivalent de la variable d'une fonction).

- Les variables locales (variable_locale1, variable_locale2,...) n'existent que dans la procédure et ne peuvent être utilisées en dehors.

- Les variables globales (variable_globale1, variable_globale2,...) existent en dehors de l'exécution de la procédure et peuvent être appelées et utilisées après l'exécution.

Structures conditionnelles et itératives :

```
if condition1 then
        instructions;
elif condition2 then
        instructions;
else
        instructions;
fi;
```

```
Si condition1 est vrai alors
        faire instructions
Sinon, si condition2 est vrai alors
        faire instructions
Sinon,
        faire instructions
Fin du Si
```

elif est facultatif si vous n'avez pas de condition supplémentaire à indiquer.
else est facultatif si le programme ne doit rien faire lorsque *condition1* n'est pas vérifiée.

```
for k from a to b by p do
        instructions;
od;
```

```
Pour k allant de a à b avec un pas de p, faire
        instructions
Fin du Pour
```

- Le pas *p* peut être omis et est égal à 1 par défaut.
  Ainsi la syntaxe peut être simplifiée de cette façon : **for** *k* **from** *a* **to** *b* **do [...] od;**

- Le pas *p* peut aussi être négatif si l'itération doit être décroissante
  (dans ce cas on doit avoir a > b)

```
while condition do
        instructions;
od;
```

```
Tant que condition est vraie, faire
        instructions
Fin du Tant que
```

3. Structures de données

Séquences	
seq(expr(i),i=a..b) ex: seq(2*i, i=1..5)	séquence des termes expr(i) pour i allant de a à b 2, 4, 6, 8, 10
expr(i)$i=a..b ex: 2*i$i=1..5	même résultat qu'au dessus 2, 4, 6, 8, 10
x$n ex: x$4	répétition de x (n fois) x, x, x, x
A[i]	i-ème élément de la séquence A
A := A,e	ajoute l'élément e à la fin de la séquence A
A := e,A	ajoute l'élément e au début de la séquence A
A,B	séquence composée des séquences A et B mises bout à bout
max(A)	maximum de la séquence A
min(A)	minimum de la séquence A

Listes	
[A] ex: [2, 4, 6, 8, 10]	liste créée à partir de la séquence A
nops(L)	taille de la liste L
select(fonction,L) ex: select(x -> x>0, [1,-1,2,-3])	liste des éléments x de L qui vérifient la condition (fonction(x) = true) renvoie [1,2]
remove(fonction,L) ex: remove(x -> x>0, [1,-1,2,-3])	enlève les éléments x de L qui vérifient la condition (fonction(x) = true) renvoie [-1,-3]
member(e,L)	test d'appartenance de l'élément e à la liste L
op(L) ex: op([1,2,3])	convertit la liste L en séquence renvoie 1, 2, 3
[op(L1),op(L2)]	concaténation des listes L1 et L2
map(f,L) ex: map(f,[a,b,c,d])	applique la fonction f à tous les éléments de L renvoie [f(a),f(b),f(c),f(d)]
sort(L)	trie la liste L dans l'ordre croissant