Dans tous les cas, sachez utiliser l'aide de Maple. Notamment en plaçant le curseur dans un mot et en tapant `F1` ou `F2`

1. Linéarisation d'un polynôme trigonométrique

   ```
   combine(expr,trig);
   ```

2. Pour transformer une expression trigonométrique de x en expression rationnelle de $\tan\frac{x}{2}$

   ```
   convert(expr,tan);
   ```

   on peut remplacer `tan` par `cos` ou `sin`. Ces transformations sont nécessaires si on veut imposer à Maple un changement de variable particulier.

3. Simplification

   ```
   simplify(expr,symbolic);
   ```

   L'effet de `symbolic` est de ne pas chercher à vérifier la validité des transformations. Par exemple, une simplification de $\sqrt{x^2}$ renverra x sans fonction particulière relative au signe de x.

4. Décomposition en éléments simples d'une expression qui est une fraction rationnelle

   ```
   convert(expr,parfrac,x);
   ```

5. Rassembler des facteurs

   ```
   collect(expr,[x,y,z]);
   ```

   où `expr` est une expression en `x`,`y`,`z`. Essaie de renvoyer une expression de la forme

   $$(\text{ somme de trucs })\,x + (\text{ somme de machins })\,y + (\text{ somme de choses })\,z$$

6. Limite d'une expression de x en a.

   ```
   limit(expr,x=a);
   ```

   On peut prendre `x=infinity` ou `x=-infinity`

7. Développement limité d'une expression de x en a

   ```
   taylor(expr,x=a,n)
   ```

   Attention ! n n'est pas forcément l'ordre du développement renvoyé, c'est l'ordre avec lequel les développements élémentaires nécessaires au calcul sont effectués.
   Ce paramètre est facultatif (6 par défaut). Le reste d'un développement limité donné par Maple est en $O(x-a)^k$.

8. Autres développements d'une expression en a

   ```
   series(expr,x=a,n);
   ```

9. Intégrale d'une expression de x entre des bornes a et b.

   ```
   int(expr,x=a..b);
   ```

   Maple essaie de calculer une intégrale en utilisant ses algorithmes. Si on omet le `=a..b`, (en laissant `x`) la fonction renvoie (si possible) une primitive.
   Si on ne veut pas des algorithmes de Maple, on peut utiliser la version *inerte* de la fonction `Int(expr,x=a..b)`

10. Somme d'une expression en i de m à n

    ```
    sum(expr,i=m..n);
    ```

    Cette fonction essaie de calculer une expression de la somme avec des fonctions usuelles (pour Maple ! pas forcément pour nous.). La problématique est à peu près la même que pour une intégrale. Si on veut additioner comme avec une boucle `for`, il faut utiliser `add(expr,i=m..n)`. Lors de l'exécution, n doit s'évaluer à une valeur numérique. La forme *inerte* `Sum(expr,i=m..n)` peut aussi être utile. Vous pouvez aussi utiliser `i=m..infinity` pour la somme d'une série (programme spé).

11. Produit d'une expression en i de m à n

    ```
    product(expr,i=m..n);
    ```

    Cette fonction est l'analogue de `sum` pour le produit. Pour des multiplications comme dans une boucle `for` utiliser `mul(expr,i=m..n)`. La forme *inerte* `Product(expr,i=m..n)` peut aussi être utile.

12. Bibliothèque "student". Elle contient diverses fonctions : intégration par parties, changement de variable, ...

    ```
    with(student);
    ```

13. Intégration par parties (dans la bibliothèque `student`).

    ```
    intparts(intgr,p);
    ```

    où `intgr` est un objet Maple de type intégrale, par exemple d'une expression `expr` en x et `p` une expression en x. Le changement de variable se fera en prenant `p` comme primitive.

14. Changement de variable (dans la bibliothèque `student`).

    ```
    changevar(x=f(u),intgr,u);
    ```


Rémy Nicolai Aoralcentr

où `intgr` est un objet Maple de type intégrale, par exemple d'une expression `expr` en x. La fonction renvoie une nouvelle intégrale d'une expression de `u`. On peut aussi faire le changement de variable dans l'autre sens.

```
changevar(f(x)=u,intgr,u)
```

15. Penser à utiliser `sinh` et `cosh` pour les sinus et cosinus hyperboliques.
16. Résolution d'équations.

    ```
    solve(Eq,Inc);
    ```

    où `Eq` est un *ensemble* d'équations et `Inc` est un *ensemble* d'inconnues. La fonction renvoie un ensemble d'égalités qui peut être utilisé tel que dans la fonction `subs`.
17. Représentation graphique de fonctions.

    ```
    plot([ex1,ex2,ex3],x=xm..xM,y=ym..yM,color=[blue,yellow,red]);
    ```

    où `ex1`, ... sont des expressions en `x`.
18. Dérivations. Pour une fonction utiliser `D(f)`. Pour une expression de `x`, utiliser `diff(expr,x`.

Rémy Nicolai Aoralcentr