So you want to use Google Test ?

**The Basics :**

Google test vous permet d’examiner le comportement de votre code dans un contexte prédéfini, en vous concentrant sur une petite portion du code à la fois (p.ex. tester une méthode). Ceci évite donc de devoir compiler tout un programme, et de devoir gérer des paramètres qui n’ont rien à voir avec le code que vous voulez tester. C’est l’intérêt des *unit tests*.

Pour ce faire, on crée un fichier (p.ex. *test.cpp*) dans lequel on définit tous les tests qu’on aimerait faire passer à notre code. Google test s’occupe ensuite de créer un main bidon et de lancer les tests. Il vous donne ensuite les résultats dans le terminal. On peut tout mettre dans un seul fichier sans soucis parce que gtest nous permet de spécifier quels tests on aimerait lancer, et lesquels on ne voudrait pas lancer. (Donc si vous ne vous intéressez qu’à tester votre fonction *readDNA()*, vous n’allez pas lancer tous les tests pour la classe PWM !)

**Voici comment on écrit un test**:

>>> TEST(testCase, testName) <<  ‘un message qui sera affiché lors d’un échec du test’

>>> {

>>> …test code…

>>> }

TEST est une macro définie par gtest. *testCase*, et *testName* sont juste des noms qui vous permettent de structurer vos tests selon des cas (p.ex. une classe du code peut être un cas) et des tests proprement dits (p.ex. les méthodes de la classe).

Ensuite pour le test code, vous utiliserez des macros définies par gtest. Voici un exemple :

>>> ASSERT\_TRUE(readDNAFromFile(« nom d’un fichier »));

ASSERT\_TRUE va vérifier que l’expression passée en argument est vraie (dans ce cas, l’expression est une fonction, donc il faut qu’elle retourne la valeur vrai).

La liste des macros peut être consultée dans cette référence : <https://github.com/google/googletest/blob/master/googletest/docs/Primer.md>

À noter que toutes les macros existent à double. Les versions ASSERT() quittent la fonction dans le cas d’une erreur, alors que les versions EXPECT() continuent les autres tests même dans le cas d’une erreur. Ainsi on préférera les macros EXPECT() à moins que ça ne fasse pas sens de continuer les tests après une erreur (p.ex. si on test l’ouverture du fichier et que la fonction est même pas capable de créer son fstream, ça ne sert à rien de continuer, il faut d’abord régler ce problème là !)

**Les fixtures**

Si vous devez tester une classe, et que cette classe est embêtante à initialiser (il y a une tonne de paramètres, faut ouvrir des fichiers, tout configurer etc.) gtest vous donne un outil pour éviter de faire ça à chaque fois !

Une fixture est en gros un objet qui contient tous les variables, infos, objets, etc. nécessaires pour lancer vos tests. Vous pouvez ensuite lancez tous les tests que vous voulez avec ce framework, sans vous souciez de redéclarer des variables etc.

**Comment ça marche ce truc ?**

Vous commencez par créer une classe qui dérive de *::testing ::Test*. Donc par exemple :

>>> monTest : public ::testing ::Test

>>> {

>>> public:

>>> …

>>> }

Dans cette classe, vous déclarez tous les objets que vous voulez. Notez que c’est important de tout mettre ***publique*** puisqu’on aimerait y avoir accès pour les tests.

S’il y a besoin vous pouvez déclarer un constructeur ou une méthode *SetUp()* qui serait appelé avant les tests. Donc typiquement si vous créez un objet PWM avec des probabilités, et que vous voulez ensuite le covertir en une matrice avec des logs, il faut le faire soit dans le constructeur soit dans la méthode *SetUp()*.

Si besoin vous pouvez aussi déclarer un déstructeur ou un méthode *TearDown()* qui sont appelés à la fin des tests (par exemple si vous utilisez des pointeurs !!)

Maintenant pour écrire vos tests vous utilisez simplement la macro TEST \_F :

>>> TEST\_F(monTest, test01)

>>> {

>>> …

>>>}

>>> TEST\_F(monTest, test02)

etc.

Voilà, si vous voulez avoir plus de détails, suivez ce lien : <https://github.com/google/googletest/blob/master/googletest/docs/Primer.md>

(et si vous en voulez encore plus : <https://github.com/google/googletest/blob/master/googletest/docs/AdvancedGuide.md>,

et voici des exemples : <https://github.com/google/googletest/blob/master/googletest/docs/Samples.md>)