1. Intégration – Méthode des trapèzes

On souhaite calculer la surface située entre 2 bornes d'une fonction continue, en l'occurrence la fonction de densité de la loi normale centrée (moyenne nulle) et réduite (écart type = 1) qui s'écrit :

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

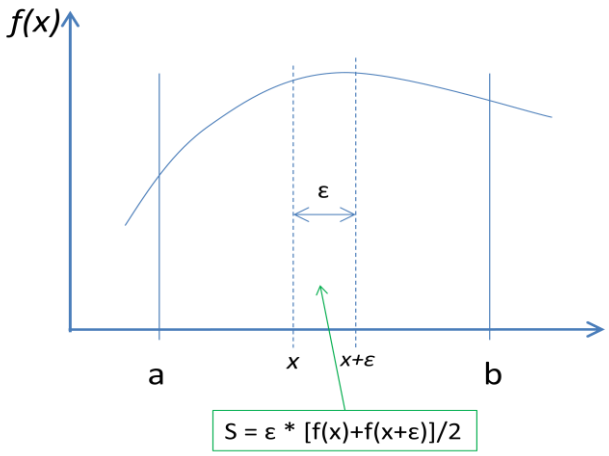
Ecrivez votre programme sous forme de fonction que vous intégrez dans une unité, elle prend en entrée 2 réels, et renvoie un réel.

```
function trapeze_normale(a,b : double) : double ;
```

Description de l'algorithme :

L'idée de la méthode des trapèzes consiste à subdiviser l'espace allant de « a » à « b » en une série de trapèzes de largeur ε (fixons ε = `0.00001` dans notre implémentation). On sait que la surface d'un trapèze est égale à S = ε * [f(x) + f(x + ε)]/2. Il suffit d'additionner les surfaces intermédiaires pour obtenir la surface totale correspondant à une intégrale.

Schématiquement, nous aurions ceci :



Remarques :

- Assurez-vous que (b >= a), la fonction renvoie la valeur 0 dans le cas contraire.
- Vous devez intégrer votre fonction dans une unité. Dans le programme principal (faites une application console pour simplifier), vous demanderez à l'utilisateur de saisir « a » et « b », vous faites alors appel à votre fonction pour afficher les résultats.
- Vous pouvez utiliser le tandem « Python + librairie ' Scipy ' » pour vérifier l'exactitude de vos calculs (http://tutoriels-data-mining.blogspot.com/2017/04/probabilites-et-quantiles-sous-

excel-r.html). Votre fonction doit renvoyer un résultat équivalent à « `stats.norm.cdf(b) - stats.norm.cdf(a)` »

2. Fonction de répartition de la loi normale

On souhaite enrichir notre unité en rajoutant la fonction de répartition de la loi normale centrée et réduite. Schématiquement, elle calculerait la surface allant de] –∞, q] (« q » est le paramètre d'entrée de votre fonction).

Nous pouvons exploiter la solution développée dans l'exercice précédent, sachant que :

- La surface totale allant de] –∞, +∞[est égale à 1.
- La fonction de densité est symétrique autour de 0.
- La surface allant de] –∞, 0] est égale à 0.5

Voici la signature :

```
function fnorm(q : double) : double ;
```

Après avoir implémenté la fonction dans votre unité, modifiez le programme principal pour que seul le paramètre « q » soit demandé. Faites appel à votre fonction par la suite.

Remarque : Votre programme devrait fournir un résultat identique à « `stats.norm.cdf(q)` ».

3. Programmation et importation d'une DLL

Nous souhaitons déployer notre fonction « fnorm » sous la forme d'une libraire compilée DLL (https://www.youtube.com/watch?v=2ePenwwJqIw).

Créez un projet « Bibliothèque » sous Lazarus. Importez l'unité développée précédemment. Exportez la seule fonction « fnorm ». Compilez votre projet, vérifiez que le fichier « .dll » a bien été généré.

Créez ensuite un nouveau projet « console ». Importez « fnorm » de la DLL. Demandez à l'utilisateur de saisir une valeur « q » pour vérifier son bon fonctionnement.

4. Importation de la DLL sous Python

Elaborez un programme Python (.py) au sein duquel vous importez la librairie, puis la fonction « fnorm ». Vérifiez son bon fonctionnement en lui soumettant plusieurs valeurs exemples de « q ». Comparez vos résultats avec ceux de « `stats.norm.cdf(q)` » du package « Scipy ».

5. Comparaison des performances – Implémentation Python

Nous souhaitons comparer la rapidité d'exécution de « fnorm » issue de notre DLL, une librairie compilée rappelons-le, avec une implémentation « pure » Python.

Programmez sous Python la fonction « `fnorm_python(q)` » équivalente à celle développée sous Lazarus. Utilisez les mêmes structures de code pour que les deux stratégies soient comparables (évitez d'utiliser des astuces propres au langage). Testez les mêmes valeurs de « q », vérifiez dans un premier temps que vous obtenez les mêmes résultats.

Dans un deuxième temps, avec la commande « magique » %timeit de la console IPython, mesurez et confrontez leur temps d'exécution. Que constatez-vous ? (le code compilé est 30 fois plus rapide sur la machine).

Remarque : Apparemment, il est possible de compiler des implémentations Python avec des modifications mineures dans le code (http://tutoriels-data-mining.blogspot.com/2019/11/jit-compilation-sous-python.html). Adaptez votre programme en conséquence, refaites les tests de performances. Qu'observez-vous cette fois-ci ? (la fonction Python « compilée » est autrement plus performante…)