

```
import UIKit //permet de construire et gérer l'affichage graphique de l'interface
utilisateur pour iOS
```

```
import CoreMotion //permet de récupérer les valeurs du gyroscope, de
l'accéléromètre et du podomètre pour les utiliser dans l'application
```

```
import CoreLocation //permet d'obtenir les coordonnées géographiques de
localisation et d'orientation
```

```
class ViewController: UIViewController { //permet l'affichage des données
```

```
//importation des label
```

```
@IBOutlet weak var ltr: UILabel! //trend
```

```
@IBOutlet weak var lpl: UILabel! //plunge
```

```
@IBOutlet weak var lpldir: UILabel! //PlungeDirection
```

```
@IBOutlet weak var pstr: UILabel! //strike
```

```
@IBOutlet weak var pdip: UILabel! //Dip
```

```
@IBOutlet weak var pdipdir: UILabel! //DipDirection
```

```
@IBOutlet weak var `switch`: UISwitch! //importation du bouton switch
```

```
@IBAction func controle(_ sender: Any) {
```

```
    if switch.isOn {
```

```
        mesure()
```

```
        switch.setOn(false, animated: true)
```

```
    } else {
```

```
        mesure()
```

```
        switch.setOn(true, animated: true)
```

```
    }
```

```
} //affihe la mesure quand le switch est activé
```

```
let pi = 3.14159
```

```
var ori = CMMotionManager() // pour démarrer l'objet ori qui récupère les
valeurs xTrueNorthZVertical
```

```
var pos = CLLocationManager() //bibliothèque complémentaire au CoreMotion
permettant de récupérer les valeurs
```

```

var cnl = Double() // premiere composante de la ligne (suivant North)
var cel = Double() // deuxieme composante de la ligne (suivant East)
var cdl = Double() //troisieme composante de la ligne (suivant Down)

var cnpp = Double() // premiere composante du pole plan (suivant North)
var cepp = Double() // deuxieme composante du pole plan (suivant East)
var cdpp = Double() // troisieme composante du pole plan (suivant Down)

var vartrend = Double() //Double apporte une double précision a l'affichage du
nombre a virgule

var varstrike = Double()

override func viewDidLoad() {
    super.viewDidLoad() // fait les setup en plus après avoir chargé la vue

    pos.startUpdatingHeading() //commence a generer des updates qui donnent le
cap de l'utilisateur

    mesure()
} // ferme override

func mesure() { // donne trend et plonge du long bord du telephone
    ori.deviceMotionUpdateInterval = 0.5 //tout les combien de temps la mesure
s'actualise
    ori.startDeviceMotionUpdates(using:
CMAttitudeReferenceFrame.xTrueNorthZVertical, to: OperationQueue.current!){
//démontre les updates de xTrueNorthZVertical qui prend Z axe pour vertical et X axe
pointe vers le true north

    (data, error) in
    if let true_data = data{
        // POUR LA LIGNE
        //utilise la matrice de rotation entre le systeme de coordonnées North-East-Down
et celui du telephone, en relation avec le tangage, le roulis et le lacet

        self.cnl = true_data.attitude.rotationMatrix.m21 //pour la ligne north
        self.cel = -true_data.attitude.rotationMatrix.m22 //pour la ligne east
        self.cdl = -true_data.attitude.rotationMatrix.m23 //pour la ligne down
        //attention aux moins dûs au sens des axes

        let coordl = self.cart2sph(cn: self.cnl, ce: self.cel, cd: self.cdl)
        // utilise la fonction cart2sph (voir plus bas)
        //utilise les coordonnées après passage de la matrice de rotation

```

```

self.vartrend = ((coordl.tr + 2*self.pi).truncatingRemainder(dividingBy:
2*self.pi))
// affichage ltr entre 0 et 180 exclus, on ajoute 2 pi pour etre toujours +
//truncatingRemainder renvoie le reste de la valeur divisé par la valeur donnée

if (self.vartrend < self.pi) { //cas inférieur à pi
    self.ltr.text = "\((Int(self.vartrend*180/self.pi))"
    // Int pour avoir la partie entière // *180/pi pour passer des radians en
degrés

} else {
    self.ltr.text = "\((Int((self.vartrend - self.pi)*180/self.pi))"
    //reste des cas : on soustrait pi
}

// affichage du plunge
self.lpl.text = "\((Int(abs(coordl.pl*180/self.pi)))" //abs pour avoir la valeur
absolue pour ne pas avoir les valeurs en négatif dans l'autre sens

// affichage du lpdir selon l'angle du trend (en multiple de pi)
if (coordl.pl >= 0) {
    if (self.vartrend < self.pi/8 || self.vartrend >= 15*self.pi/8) {
        self.lpdir.text = "N"
    } else if (self.pi/8 < self.vartrend && self.vartrend <= 3*self.pi/8) {
        self.lpdir.text = "NE"
    } else if (3*self.pi/8 < self.vartrend && self.vartrend <= 5*self.pi/8) {
        self.lpdir.text = "E"
    } else if (5*self.pi/8 < self.vartrend && self.vartrend <= 7*self.pi/8) {
        self.lpdir.text = "SE"
    } else if (7*self.pi/8 < self.vartrend && self.vartrend <= 9*self.pi/8) {
        self.lpdir.text = "S"
    } else if (9*self.pi/8 < self.vartrend && self.vartrend <= 11*self.pi/8) {
        self.lpdir.text = "SW"
    } else if (11*self.pi/8 < self.vartrend && self.vartrend <= 13*self.pi/8) {
        self.lpdir.text = "W"
    } else if (13*self.pi/8 < self.vartrend && self.vartrend <= 15*self.pi/8) {
        self.lpdir.text = "NW"
    }
} else {
    if (self.vartrend < self.pi/8 || self.vartrend >= 15*self.pi/8) {
        self.lpdir.text = "S"
    } else if (self.pi/8 < self.vartrend && self.vartrend <= 3*self.pi/8) {
        self.lpdir.text = "SW"
    } else if (3*self.pi/8 < self.vartrend && self.vartrend <= 5*self.pi/8) {
        self.lpdir.text = "W"
    } else if (5*self.pi/8 < self.vartrend && self.vartrend <= 7*self.pi/8) {
        self.lpdir.text = "NW"
    } else if (7*self.pi/8 < self.vartrend && self.vartrend <= 9*self.pi/8) {
        self.lpdir.text = "N"
    }
}

```

```

    } else if (9*self.pi/8 < self.vartrend && self.vartrend <= 11*self.pi/8) {
        self.lpdir.text = "NE"
    } else if (11*self.pi/8 < self.vartrend && self.vartrend <= 13*self.pi/8) {
        self.lpdir.text = "E"
    } else if (13*self.pi/8 < self.vartrend && self.vartrend <= 15*self.pi/8) {
        self.lpdir.text = "SE"
    }
}

```

// POUR LE PLAN

//utilise la matrice de rotation entre le systeme de coordonnées North-East-Down
et celui du telephone, en relation avec le tangage, le roulis et le lacet

```

self.cnpp = true_data.attitude.rotationMatrix.m31 //pour le pole plan north
self.cepp = -true_data.attitude.rotationMatrix.m32 //pour le pole plan east
self.cdpp = -true_data.attitude.rotationMatrix.m33 //pour le pole plan down

```

```

let coordp = self.cart2sph(cn: self.cnpp, ce: self.cepp, cd: self.cdpp)
// utilise la fonction cart2sph (voir plus bas)
//utilise les coordonnées après passage de la matrice de rotation

```

```

self.varstrike = ((coordp.tr + 2*self.pi +
self.pi/2).truncatingRemainder(dividingBy: 2*self.pi))
// strike = trend du pole + pi/2 et en plus on 2pi pour etre toujours +
//truncatingRemainder renvoie le reste de la valeur divisé par la valeur donnée

```

```

// affichage pstr entre 0 et 180 exclus
if (self.varstrike < self.pi) { //cas inférieur à pi
    self.pstr.text = "\((Int(self.varstrike*180/self.pi))"
    // Int pour avoir la partie entière // *180/pi pour passer des radians en
degrés

```

```

} else {
    self.pstr.text = "\((Int((self.varstrike - self.pi)*180/self.pi))"
    //reste des cas : on soustrait pi
}

```

```

// affichage du dip
self.pdip.text = "\((Int((self.pi/2 - abs(coordp.pl))*180/self.pi))"
// dip = 90 - plunge du pole

```

```

// affichage de pdipdir selon l'angle du strike (en multiple de pi)
if (coordp.dip >= 0) {
if (self.varstrike < self.pi/8 || self.varstrike >= 15*self.pi/8) {
    self.pdipdir.text = "E"
} else if (self.pi/8 < self.varstrike && self.varstrike <= 3*self.pi/8) {
    self.pdipdir.text = "SE"
} else if (3*self.pi/8 < self.varstrike && self.varstrike <= 5*self.pi/8) {

```

```

        self.pdipdir.text = "S"
    } else if (5*self.pi/8 < self.varstrike && self.varstrike <= 7*self.pi/8) {
        self.pdipdir.text = "SW"
    } else if (7*self.pi/8 < self.varstrike && self.varstrike <= 9*self.pi/8) {
        self.pdipdir.text = "W"
    } else if (9*self.pi/8 < self.varstrike && self.varstrike <= 11*self.pi/8) {
        self.pdipdir.text = "NW"
    } else if (11*self.pi/8 < self.varstrike && self.varstrike <= 13*self.pi/8) {
        self.pdipdir.text = "N"
    } else if (13*self.pi/8 < self.varstrike && self.varstrike <= 15*self.pi/8) {
        self.pdipdir.text = "NE"
    }
} else {
    if (self.varstrike < self.pi/8 || self.varstrike >= 15*self.pi/8) {
        self.pdipdir.text = "W"
    } else if (self.pi/8 < self.varstrike && self.varstrike <= 3*self.pi/8) {
        self.pdipdir.text = "NW"
    } else if (3*self.pi/8 < self.varstrike && self.varstrike <= 5*self.pi/8) {
        self.pdipdir.text = "N"
    } else if (5*self.pi/8 < self.varstrike && self.varstrike <= 7*self.pi/8) {
        self.pdipdir.text = "NE"
    } else if (7*self.pi/8 < self.varstrike && self.varstrike <= 9*self.pi/8) {
        self.pdipdir.text = "E"
    } else if (9*self.pi/8 < self.varstrike && self.varstrike <= 11*self.pi/8) {
        self.pdipdir.text = "SE"
    } else if (11*self.pi/8 < self.varstrike && self.varstrike <= 13*self.pi/8) {
        self.pdipdir.text = "S"
    } else if (13*self.pi/8 < self.varstrike && self.varstrike <= 15*self.pi/8) {
        self.pdipdir.text = "SW"
    } // ferme if varstrike
} // ferme true_data
} // ferme startupdate
} // ferme mesure

```

```

func cart2sph(cn: Double, ce: Double, cd: Double) -> (tr: Double, pl: Double){
//transforme cn, ce, cd (suivant le système de coordonnées North-East-Down) en trend
&
plunge
    let pl = asin(cd)
    var tr = Double()
    if (cn == 0) { // cas particulier ou cn = 0 (ligne est-ouest)
        if (ce < 0) {
            tr = 3*self.pi/2
        } else {
            tr = self.pi/2
        }
    }
    } else {
        tr = atan(ce/cn) // cas general
        if (cn < 0) {

```

```

        tr = (tr + self.pi).truncatingRemainder(dividingBy: 2*self.pi)
        //truncatingRemainder renvoie le reste de la valeur divisé par la valeur donnée
    } else {
        tr = tr.truncatingRemainder(dividingBy: 2*self.pi)
    }
}
return (tr, pl)
} // ferme cart2sph

@objc func changement(c_switch: UISwitch) {    //gère le bouton switch
    if c_switch.isOn {    //affiche la mesure
        mesure()
    } else {    //mesure en continue
        mesure()
    }
}

} // ferme ViewController

```