Ensoleilement

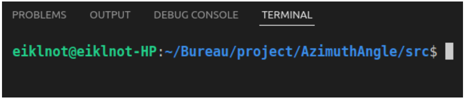
BEN MARZOUK Mohamed

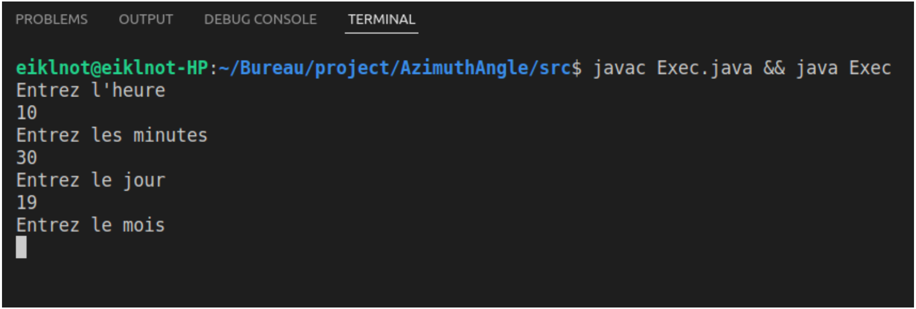
Université du Havre

Plan

- Position du Soleil
- L'ombre la portée
- La quantité d'ensoleilement reçus par une surface

Execution du programme





Execution du programme

```
eiklnot@eiklnot-HP:~/Bu
Entrez l'heure
10
Entrez les minutes
0
Entrez le jour
21
Entrez le mois
6
Entrez l'année
2022
Entrez la latitude
49
Entrez la longitude
2,19
Entrez le UTC
0
```

Execution du programme

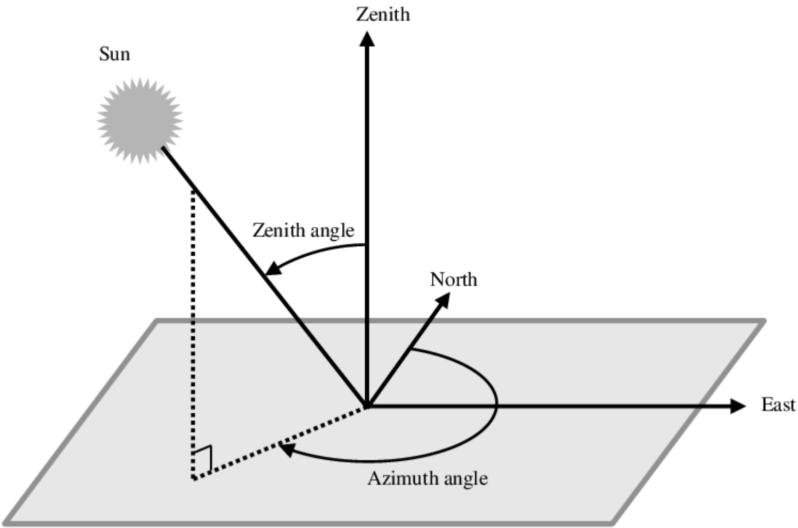
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
L'angle Azimuth (A) est : 140.3591752672196
 ********************************************
L'angle de l'elevation (alpha) du soleil est : 59.695731093881164
 ********************************************
L'angle de declinaison (delta) du soleil est : 23.369707423042353
********************************************
L'angle horaire (hour) du soleil est : -20.529191071531272
********************************************
la longueur de l'ombre (L)=la hauteur de l'objet (C) x cotang(l'angle de l'elevation (alpha))
*********************************************
L=Cx0.5844527713006744
********************************************
S'(la pente (p),Angle azimut à la normal (An)) = derivé de l'angle incident (i')xcos(angle incident (i) )
*********************************************
cos(i)=sin(p) x cos(alpha) x cos(A-An) + cos(p) x sin(alpha)
*********************************************
cos(i) = sin(p) x cos(140.3591752672196 - An) +cos(p) x 0.8633579576394883
*********************************************
i = acos(sin(p) x cos(alpha) x cos(A-An) + cos(p) x sin(alpha))
********************************************
i = acos(sin(p) x cos(140.3591752672196 - An) +cos(p) x 0.8633579576394883 )
*********************************************
i' = 1/Math.sqrt(1-i^2)
*********************************************
i' = 1/Math.sqrt(1- (acos(sin(p) x cos(140.3591752672196 - An) +cos(p) x 0.8633579576394883 ))^2)
*********************************************
S'(p,An) =1/Math.sqrt(1- (acos(sin(p) x cos(140.3591752672196 - An) +cos(p) x 0.8633579576394883 ))^2) x (sin(p
*********************************************

eiklnot@eiklnot-HP:~/Bureau/project/AzimuthAngle/src$
```

Position du Soleil

- L'angle de l'azimut
- L'angle de l'élévation

L'angle de l'Azimut



L'angle de l'Azimut

- $$Si \;\; H < 0 : A = cos^{-1}(\frac{sin(\delta)cos(L) - cos(\delta)sin(L)cos(H)}{cos(\alpha)}) \qquad (1)$$

```
Math.acos((Math.sin(delta * (Math.PI / 180)) * Math.cos(latitude * (Math.PI / 180))
        - Math.cos(delta * (Math.PI / 180)) * Math.sin(latitude * (Math.PI / 180))
                    * Math.cos(hour1 * (Math.PI / 180)))
        / Math.cos(alpha * (Math.PI / 180))) * 180 / Math.PI;
```

- $$Si \;\; H \geq 0 : A = 360 - cos^{-1}(\frac{sin(\delta)cos(L) - cos(\delta)sin(L)cos(H)}{cos(\alpha)}) \qquad (2)$$

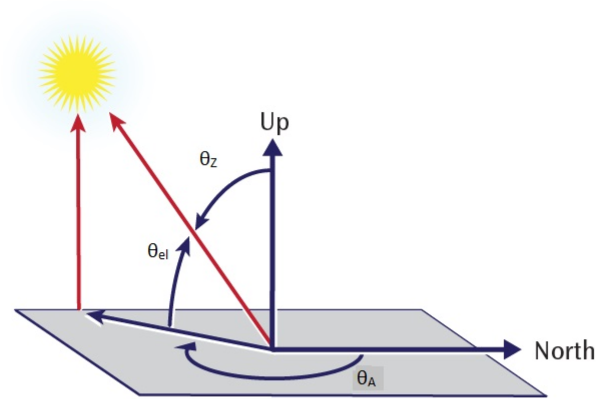
```
360 - Math.acos((Math.sin(delta * (Math.PI / 180)) * Math.cos(latitude * (Math.PI / 180))
        - Math.cos(delta * (Math.PI / 180)) * Math.sin(latitude * (Math.PI / 180))
                    * Math.cos(hour1 * (Math.PI / 180)))
        / Math.cos(alpha * (Math.PI / 180))) * 180 / Math.PI;
```

- La latitude L
- L'angle de déclinaison δ
- L'angle horaire H
- L'angle d'élévation α

L'angle de l'Azimut

```java
public class AzimuthAngle {
	private DeclinationAngle declinationAngle;
	private double latitude;
	private ElevationAngle elevationAngle;
	private HourAngle hourAngle;

	// Constructeur de la class azimuth
	public AzimuthAngle(HourAngle hourAngle,ElevationAngle elevationAngle, DeclinationAngle declinationAngle, double latitude) {
		this.declinationAngle = declinationAngle;
		this.latitude = latitude;
		this.elevationAngle= elevationAngle;
		this.hourAngle =hourAngle;
	}

	// cette methode get_value nous calcule l'angle azimuth
	public double get_value() {
		double delta = declinationAngle.get_value();
		double alpha = elevationAngle.get_value();
		double hour1 = hourAngle.get_value();

		if (hour1 >= 0) {

			return 360 - Math.acos((Math.sin(delta * (Math.PI / 180)) * Math.cos(latitude * (Math.PI / 180))
					- Math.cos(delta * (Math.PI / 180)) * Math.sin(latitude * (Math.PI / 180))
							* Math.cos(hour1 * (Math.PI / 180)))
					/ Math.cos(alpha * (Math.PI / 180))) * 180 / Math.PI;
		} else {
			return Math.acos((Math.sin(delta * (Math.PI / 180)) * Math.cos(latitude * (Math.PI / 180))
					- Math.cos(delta * (Math.PI / 180)) * Math.sin(latitude * (Math.PI / 180))
							* Math.cos(hour1 * (Math.PI / 180)))
					/ Math.cos(alpha * (Math.PI / 180))) * 180 / Math.PI;
		}
	}

	public String print_value(){
		return "L'angle Azimuth (A) est : " + get_value()+"\n********************************************";
	}
}
```

L'angle d'Élévation



θ_{el} = elevation angle, measured up from horizon

θ_Z = zenith angle, measured from vertical

θ_A = azimuth angle, measured from North

L'angle d'Élévation

$$\alpha = sin^{-1}(sin(L)sin(\delta) - cos(L)cos(\delta)cos(H)) \tag{3}$$

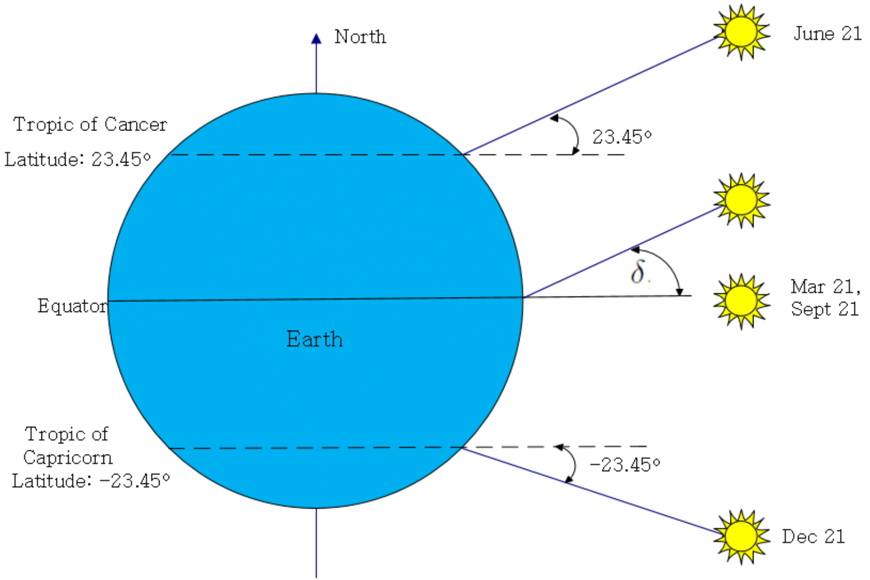
```
(Math.asin(Math.sin(latitude * Math.PI / 180) * Math.sin(delta * Math.PI / 180)
 + Math.cos(latitude * Math.PI / 180) * Math.cos(delta * Math.PI / 180)
         * Math.cos(hour1 * Math.PI / 180)))
 * 180 / Math.PI;
```

- La latitude L
- 'angle de déclinaison δ
- L'angle horaire H

L'angle d'ÉLévation

```java
public class ElevationAngle {

    private double latitude;
    private DeclinationAngle declinationAngle;
    private HourAngle hourAngle;

    public ElevationAngle(HourAngle hourAngle, DeclinationAngle declinationAngle, double latitude) {
        this.latitude = latitude;
        this.declinationAngle = declinationAngle;
        this.hourAngle = hourAngle;
    }

    public double get_value() {
        double delta = declinationAngle.get_value();
        double hour1 = hourAngle.get_value();

        return (Math.asin(Math.sin(latitude * Math.PI / 180) * Math.sin(delta * Math.PI / 180)
                + Math.cos(latitude * Math.PI / 180) * Math.cos(delta * Math.PI / 180)
                        * Math.cos(hour1 * Math.PI / 180)))
                * 180 / Math.PI;
    }

    public String print_value() {
        return "L'angle de l'elevation (alpha) du soleil est : " + get_value()+"\n*****************
    }

}
```

L’angle de Déclinaison



L'angle de Déclinaison

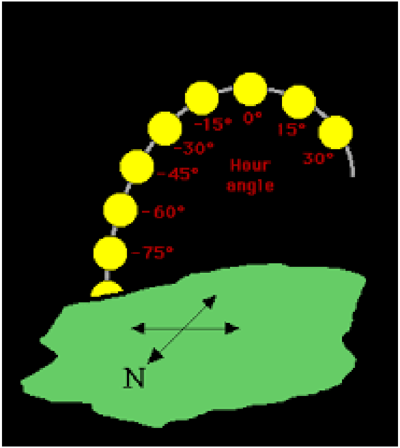
$$\delta = -23.44 \cos\left(\frac{360}{365}(d - 10)\right) \tag{4}$$

```
(-23.44 * Math.cos(((double)(360)/(double)(365) * (numberOfDays + 9))* Math.PI / 180));
```

d : le nombre de jours entre le premier jour de l'année et le jour du mois entrée.

```java
//Dans cet class j'importe le calendrier grégorien
//afin de calculer le nombre de jours entre la date demandé et le premier jour de l'année demandé
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.concurrent.TimeUnit;

public class DeclinationAngle {
    public int numberOfDays;

    // Contructeur de la class DeclinationAngle
    public DeclinationAngle(int jour, int mois, int annee) {
        GregorianCalendar date = new GregorianCalendar(annee, mois, jour);
        GregorianCalendar firstDay = new GregorianCalendar(annee, month: 1, dayOfMonth: 1);

        this.numberOfDays = daysBetween(firstDay.getTime(), date.getTime());
    }
    // cet equation nous permet de calculer entre la date demandé et le premier jour
    // de l'année demandé
    public int daysBetween(Date d1, Date d2) {
        long diff = d2.getTime() - d1.getTime();
        return (int) TimeUnit.DAYS.convert(diff, TimeUnit.MILLISECONDS);

    }

    // Cet method get_value nous permet de calculer l'angle de déclinaison
    public double get_value() {

        //declination Angle entre -23.44 in december and 23.44 in june
        return (-23.44 * Math.cos(((double)(360)/(double)(365) * (numberOfDays + 9))* Math.PI / 180));

    }
    public String print_value(){
        return "L'angle de declinaison (delta) du soleil est : " + get_value()+"\n*********************************************";
}
}
```

L'angle Horaire



L'angle Horaire

$$H = 15(lst - 12) \tag{5}$$

```
15 * (localSolarTime.get_value()- 12);
```

lst : temps solaire local

L'angle Horaire

```java
public class HourAngle {
    private LocalSolarTime localSolarTime;

    public HourAngle(LocalSolarTime localSolarTime) {
        this.localSolarTime = localSolarTime;
    }

    public double get_value() {

        return 15 * (localSolarTime.get_value()- 12);

    }
```

lst

$$lst = LT\frac{TC}{60} \tag{6}$$

```
localTime.get_value() + timeCorrectionFactor.get_value()/60;
```

- LT : temps local
- TC : time correction factor

lst

```java
public class LocalSolarTime {
    private LocalTime localTime;
    private TimeCorrectionFactor timeCorrectionFactor;

    public LocalSolarTime(LocalTime localTime, TimeCorrectionFactor timeCorrectionFactor){
        this.localTime= localTime;
        this.timeCorrectionFactor=timeCorrectionFactor;
    }

    public double get_value(){

        return localTime.get_value() + timeCorrectionFactor.get_value()/60;
    }

}
```

Temps local

```java
public class LocalTime {
    private int hour;
    private int minute;


    public LocalTime(int hour, int minute) {
        this.hour = hour;
        this.minute = minute;

    }

    public double get_value() {
        return ((double)hour)+(((double) minute)/60);
    }

}
```

Time correction factor

$$TC = 4(Lg - 15UTC) + EoT \tag{7}$$

```
4 * (longitude - 15 * universalTimeCoordinate) + equationOfTime.get_value();
```

- Lg : longitude
- UTC : temps universel coordonné
- EoT : equation du temps

Time correction factor

```java
public class TimeCorrectionFactor {
    private double longitude;
    private EquationOfTime equationOfTime;
    private int universalTimeCoordinate;

    public TimeCorrectionFactor(EquationOfTime equationOfTime, double longitude,
            int universalTimeCoordinate) {
        this.longitude = longitude;
        this.universalTimeCoordinate = universalTimeCoordinate;
        this.equationOfTime = equationOfTime;

    }

    public double get_value() {

        return 4 * (longitude - 15 * universalTimeCoordinate) + equationOfTime.get_value();
    }
}
```

Equation du temps

$$\text{EoT} = 229{,}18\ (0.000075+0.001868\cos B - 0.032077\sin B - 0.014615\cos 2B - 0.040849\sin 2B)$$

```
229.18 * (0.000075 + 0.001868 * Math.cos(b.get_value() )
 - 0.032077 * Math.sin(b.get_value() )
 - 0.014615 * Math.cos(2 * b.get_value() )
 - 0.040849 * Math.sin(2 * b.get_value() ));
```

Equation du temps

```java
public class EquationOfTime {
    private B b;

    public EquationOfTime(B b) {
        this.b = b;
    }

    public double get_value() {

        return 229.18 * (0.000075 + 0.001868 * Math.cos(b.get_value() )
                - 0.032077 * Math.sin(b.get_value() )
                - 0.014615 * Math.cos(2 * b.get_value() )
                - 0.040849 * Math.sin(2 * b.get_value() ));


    }
}
```

B

$$B = \frac{2\pi}{365}(d - 1 + \frac{LT - 12}{24}) \tag{8}$$

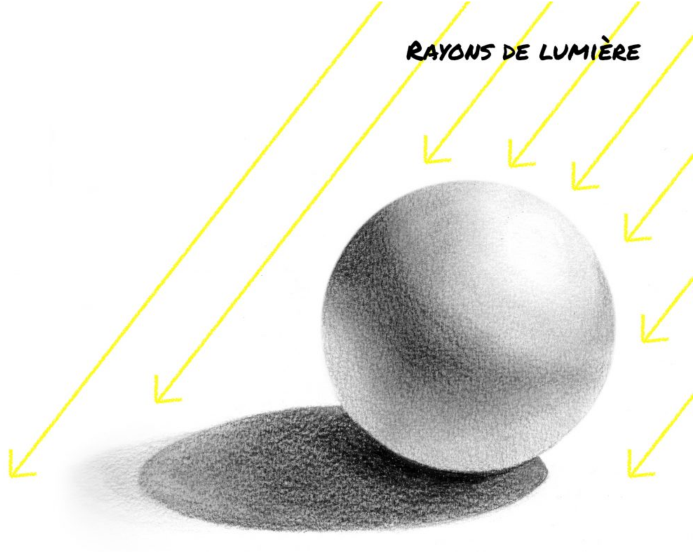
```
(2*Math.PI/365)*(numberOfDays+((localTime.get_value()-12)/24));
```

- d : le nombre de jours entre le premier jour de l'année et le jour du mois entrée.
- LT : temps local

B

```java
import java.util.GregorianCalendar;
import java.util.Date;

public class B {
    public int numberOfDays;
    private LocalTime localTime;


    public B(int hour, int minute, int jour, int mois, int annee) {
        GregorianCalendar date = new GregorianCalendar(annee, mois, jour);

        GregorianCalendar firstDay = new GregorianCalendar(annee, month: 01, dayOfMonth: 01);
        this.numberOfDays = daysBetween(firstDay.getTime(), date.getTime());
        this.localTime=new LocalTime(hour, minute);
    }

    public int daysBetween(Date d1, Date d2) {
        return (int) ((d2.getTime() - d1.getTime()) / (1000 * 60 * 60 * 24));
    }

    public double get_value() {

        return (2*Math.PI/365)*(numberOfDays+((localTime.get_value()-12)/24));
    }
}
```

Ombre porté



Ombre porte

- L : longeur de l'ombre
- C : la hauteur de l'objet
- α : l'angle de l'élévation

on a

$$\frac{C}{L} = \tan\alpha \tag{9}$$

donc

$$L = \frac{C}{\tan\alpha} \tag{10}$$

Ombre porte

```java
public class ShadowReach {
    private ElevationAngle elevationAngle;

    public ShadowReach(ElevationAngle elevationAngle) {
        this.elevationAngle = elevationAngle;
    }

    public String print_value() {
        return "la longueur de l'ombre (L)=la hauteur de l'objet (C) x cotang(l'angle de l'elevation (alpha)) \n"+
                + "L=Cx" + 1/Math.tan(elevationAngle.get_value()*Math.PI/180)+"\n********************************
    }
}
```

Énergie reçue par une surface

on a

$$S'(p, An) = i' \cos i \tag{11}$$

avec

$$\cos i = \sin p \cos \alpha \cos (A - An) + \cos p \sin \alpha \tag{12}$$

et

$$i' = \frac{1}{\sqrt{1 - i^2}} \tag{13}$$

- i : l'angle d'incidence
- p: pente de la surface etudié
- α : l'angle de l'élévation
- A : l'angle Azimut
- An :l 'angle de l'azimut à la normale

Énergie reçue par une surface

```java
public class AngleOfIncidence {

    private ElevationAngle elevationAngle;
    private AzimuthAngle azimuthAngle;

    public AngleOfIncidence(ElevationAngle elevationAngle, AzimuthAngle azimuthAngle) {
        this.elevationAngle = elevationAngle;
        this.azimuthAngle = azimuthAngle;
    }

    public String get_value() {
        return "acos(sin(p) x cos(" + azimuthAngle.get_value() + " - An) +cos(p) x "
                + Math.sin(elevationAngle.get_value() * Math.PI / 180) + " )";
    }

    public String getCosvalue() {
        return "sin(p) x cos(" + azimuthAngle.get_value() + " - An) +cos(p) x "
                + Math.sin(elevationAngle.get_value() * Math.PI / 180);
    }

    public String print_value() {
        return "S'(la pente (p),Angle azimut à la normal (An)) = derivé de l'angle incident (i')xcos(angle incident (i) ) \n"+"*****
                + print_cosval() + "\n"+"********************************************* \n"
                + print_val() + "\n"+"********************************************* \n"
                + "i' = 1/Math.sqrt(1-i^2) \n"+"********************************************* \n"
                + "i' = 1/Math.sqrt(1- (" + get_value() + ")^2) \n"+"********************************************* \n"
                + "S'(p,An) =1/Math.sqrt(1- (" + get_value() + ")^2) x (" + getCosvalue() + ")"+"\n**********************************
    }

    public String print_val() {
        return "i = acos(sin(p) x cos(alpha) x cos(A-An) + cos(p) x sin(alpha)) \n"+"********************************************* \
                + "i = " + get_value();
    }

    public String print_cosval() {
        return "cos(i)=sin(p) x cos(alpha) x cos(A-An) + cos(p) x sin(alpha) \n"+"********************************************* \n"
                + "cos(i) = " + getCosvalue();
    }
}
```

SolarParameters

```java
public class SolarParameters {

    public B b;
    public EquationOfTime equationOfTime;
    public AzimuthAngle azimuthAngle;
    public ElevationAngle elevationAngle;
    public DeclinationAngle declinationAngle;
    public HourAngle hourAngle;
    public LocalSolarTime localSolarTime;
    public LocalTime localTime;
    public TimeCorrectionFactor timeCorrectionFactor;
    public AngleOfIncidence angleOfIncidence;
    public ShadowReach shadowReach;

    public SolarParameters(int hour, int minute, int jour, int mois, int annee, double latitude, double longitude,
            int universalTimeCoordinate) {
        this.b = new B(hour, minute, jour, mois, annee);
        this.localTime = new LocalTime(hour, minute);
        this.declinationAngle = new DeclinationAngle(jour, mois, annee);
        this.equationOfTime = new EquationOfTime(b);
        this.timeCorrectionFactor = new TimeCorrectionFactor(equationOfTime, longitude, universalTimeCoordinate);
        this.localSolarTime = new LocalSolarTime(localTime, timeCorrectionFactor);
        this.hourAngle = new HourAngle(localSolarTime);
        this.elevationAngle = new ElevationAngle(hourAngle, declinationAngle, latitude);
        this.azimuthAngle = new AzimuthAngle(hourAngle, elevationAngle, declinationAngle, latitude);
        this.angleOfIncidence = new AngleOfIncidence(elevationAngle, azimuthAngle);
        this.shadowReach=new ShadowReach(elevationAngle);
    }

}
```

Class Exec

```java
import java.util.Scanner;

public class Exec {
    Run | Debug
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            int hour;
            int minute;
            int jour;
            int mois;
            int annee;
            double latitude;
            double longitude;
            int universalTimeCoordinate;

            System.out.println(x: "Entrez l'heure");
            hour = scanner.nextInt();
            System.out.println(x: "Entrez les minutes");
            minute = scanner.nextInt();
            System.out.println(x: "Entrez le jour");
            jour = scanner.nextInt();
            System.out.println(x: "Entrez le mois");
            mois = scanner.nextInt();
            System.out.println(x: "Entrez l'année");
            annee = scanner.nextInt();
            System.out.println(x: "Entrez la latitude");
            latitude = scanner.nextDouble();
            System.out.println(x: "Entrez la longitude");
            longitude = scanner.nextDouble();
            System.out.println(x: "Entrez le UTC");
            universalTimeCoordinate = scanner.nextInt();


            SolarParameters solarParameters = new SolarParameters(hour, minute, jour, mois, annee, latitude, longitude,
                    universalTimeCoordinate);

            System.out.println(solarParameters.azimuthAngle.print_value() + "\n"
                    + solarParameters.elevationAngle.print_value() + "\n"
                    + solarParameters.shadowReach.print_value() + "\n"
                    + solarParameters.angleOfIncidence.print_value());
```