Institut International d'Ingénierie de l'Eau et de l'Environnement
International Institute for Water and Environmental Engineering
Fondation 2IE

Optimisation multi-échelles d'un champ d'héliostat pour centrale à concentration de petite puissance

MASTER SPECIALISE EN GENIE ENERGETIQUE ET ENERGIES RENOUVELABLES

Présenté et soutenu publiquement le 14/12/2011 par Emmanuel CLERC

Travaux dirigés par : Yao AZOUMAH
Enseignant chercheur
UTER GEI, laboratoire LESEE. Burkina Faso

Travaux encadrés par :

Yao AZOUMAH,

Enseignant chercheur

Jury d'évaluation du stage :

Président : Albert SUNNU

Membres et correcteurs : Yao AZOUMAH
Henry COTTIN
Daniel YAMEUGEU

Promotion 2010/2011

CITATIONS

« Dieu est le seul être qui, pour régner, n'ait même pas besoin d'exister. »

Charles Baudelaire.

« Ce que l'on fait dans sa vie résonne dans l'éternité. »

Ridley Scott.

« Par le pouvoir de la vérité j'ai de mon vivant conquis l'univers. »

Faust.

« Notre pouvoir scientifique a dépassé notre pouvoir spirituel : nous pouvons guider des missiles mais nous détournons l'homme de sa voie. »

Martin Luther King.

REMERCIEMENTS/ DEDICACES

Ce travail est le fruit de l'association des efforts préalables d'un ensemble d'acteurs, qu'il me soit permis de leur témoigner ici toute ma gratitude. J'adresse mes plus sincères remerciements :

A Paul Giniès, directeur général de la fondation 2ie et à tout son personnel pour leur éthique et leurs efforts qui se sont concrétisés par la mise en place au Burkina Faso de formations reconnues.

A mon encadreur le Dr Yao AZOUMAH, que je ne saurais assez remercier pour la confiance qu'il m'a témoigné tout au long du stage, pour son appui scientifique, ses encouragements et plus encore pour son enthousiasme contagieux.

A Henri KOTTIN, ingénieur de recherche au LESEE, à qui je dois notamment le perfectionnement de mes connaissances en électrotechnique et pour l'aide qu'il m'a apporté dans la rédaction de mon mémoire.

A Gaye MADIEUMBE, ingénieur de recherche au LESEE, qui s'est toujours montré disponible pour me donner des conseils en mécanique ou dans l'organisation de mon travail.

A Batiste LAVIGNE, ingénieur de recherche au LESEE, et Vincent GIOAN, stagiaire au LESEE, dont le travail m'a permis de valider ou d'infirmer mes propres hypothèses. Merci également à eux pour leurs encouragements permanents.

A Jeremy ZMUDA, stagiaire au LESEE, avec qui j'ai travaillé en binôme. Merci pour son travail et ses idées, pour la réalisation du prototype, pour sa bonne humeur et sa persévérance.

A l'ensemble du personnel et stagiaires du LESEE pour leurs encouragements, leurs conseils et leur ouverture d'esprit.

A l'ensemble du corps professoral du 2ie pour la qualité des cours dispensés et le travail qu'ils ont fournis pour transmettre leurs connaissances.

Au petit personnel du 2ie (sécurité, restauration, nettoyage) qui réalise un travail exceptionnel mais pas toujours remarqué.

A tous mes amis et mes proches sans qui ma vie n'aurait certainement pas pris les mêmes tournants.

Enfin je ne remercierais jamais assez mes parents pour m'avoir donné les deux choses les plus précieuses qui soient : un amour inconditionnel et une bonne éducation.

RESUME

Dans le cadre du projet intégrateur réalisé en avril 2011, l'idée d'utiliser un système d'entraînement commun aux héliostats fût considérée comme une voie de recherche prometteuse. L'objet de ce mémoire était initialement l'optimisation multi-échelle d'un champ d'héliostat basé sur ce principe. Le concept imposait toutefois une contrainte de disposition particulière: dans le champ solaire, les héliostats devaient être alignés suivant l'axe Est-Ouest. Après une étude bibliographique sur le sujet, il fût établit qu'aucun des logiciels d'optimisation actuellement sur le marché n'était en mesure de générer des dispositions suivant cette contrainte. La création d'un logiciel adapté fis donc partie intégrante de ce mémoire (et de ce fait une analyse détaillée, la mise en équation et la programmation des facteurs de perte associés aux champs solaires pour centrales à tour). Grâce à cet outil de travail plusieurs dispositions ont été générées et sont présentées dans les résultats sous forme de scénarii qui offrent autant de choix au chef de projet. Chaque disposition est commentée en détail et de multiples facteurs (qualitatifs ou quantitatifs) sont discutés.
Enfin on évoque les perspectives, tant pour l'évolution du logiciel que pour l'amélioration des dispositions.

Mots Clés:

1 - Héliostats

2 - Optimisation multi-échelle

3 - Facteurs de pertes

4 - Logiciel

5 - Disposition

ABSTRACT

As part of the integrated project conducted in April 2011, the idea of using a drive system common to the heliostats was considered a promising avenue of research. The purpose of this study was originally multi-scale optimization of a field of heliostats based on this principle. The concept, however, imposes a specific disposition constraint: in the solar field, the heliostats must be aligned East-West. After a literature review on the subject, it was established that none of the optimization software on the market was able to generate dispositions in agreement with this constraint. The creation of a suitable software was so an integral part of this memory (and therefore a detailed analysis, implementation and programming equations of loss factors associated to field solar power plant tower). Using this software, several dispositions have been generated and are presented in the results as scenarios that offer many choices to the project leader. Each one is discussed in detail and multiple factors (qualitative or quantitative) are considered.

Finally, it discusses the prospects for both the development of software and the improvement of the dispositions.

up to 250 words

Key words:

1 - Heliostats

2 - Multi-scale optimization

3 - Loss factors

4 - Software

5 - Disposition

LISTE DES ABREVIATIONS

LESEE : Laboratoire d'énergie solaire et économie d'énergie

2ie : Institut International d'Ingénierie de l'Eau et de l'Environnement

n= numéro du jour de l'année

m= numéro du mois

j= numéro du jour dans le mois

h_{loc} = horaire local

h= heure locale

min = minutes

sec = secondes

L= latitude du lieu

l= longitude du lieu

TSM= Temps solaire moyen

TSV= Temps solaire vrai

δ = déclinaison

p_{abs} = position absolue

Eot = Equation du temps

w = angle horaire

h_{sol} = Elévation solaire

a_{sol} = Azimut solaire

DNI = Direct Normal Irradiation

LISTE DES TABLEAUX

LISTE DES FIGURES

TABLE DES MATIERES

I. INTRODUCTION

Le Burkina Faso fait face à une crise énergétique importante caractérisée par un des taux d'accès à l'électricité les plus faibles au monde (actuellement de 20%) [1]. Les populations rurales et périurbaines sont les plus touchées. Cette crise énergétique paralyse l'économie du pays. L'intermittence de l'approvisionnement freine le développement de nombreuses activités (ateliers de soudures, menuiseries…) génératrices d'emplois et donc susceptibles d'atténuer l'exode rurale. De plus la plupart des entreprises étrangères préfèrent investir leurs capitaux dans les pays du nord de l'Afrique ou l'approvisionnement est mieux sécurisé.

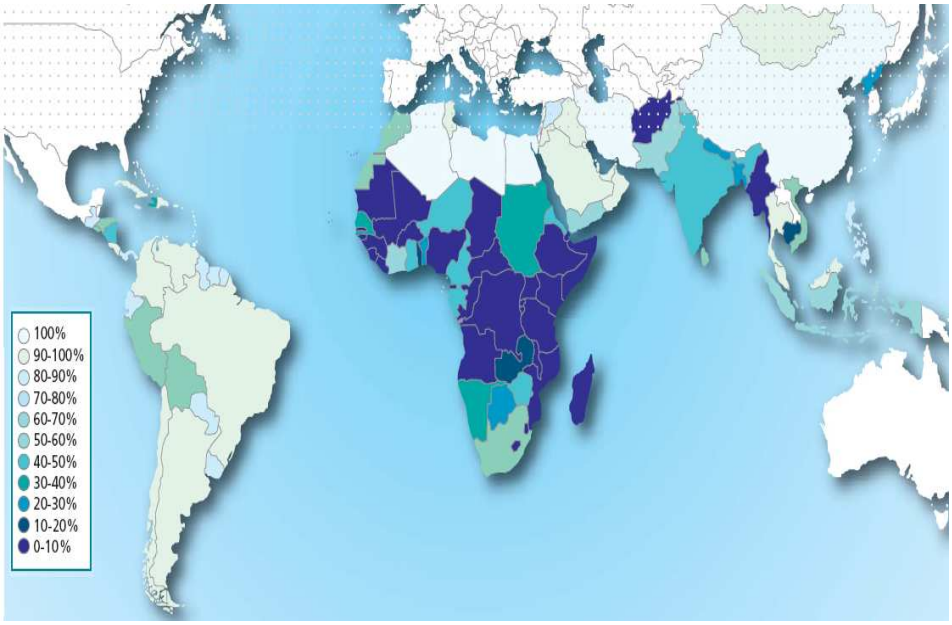


Figure 1: cartographie des taux d'accès à l'électricité dans le monde

Le Burkina Faso dispose pourtant d'une ressource qui pourrait lui permettre de développer son réseau électrique en s'affranchissant de l'augmentation du prix du pétrole : un gisement solaire exceptionnel mais largement inexploité. L'énergie solaire est en effet actuellement considérée comme non rentable comparée aux énergies fossiles. De plus la majorité des centrales solaires à concentration en fonctionnement actuel dans le monde utilisent un cycle de Rankine. Un apport d'eau est souvent nécessaire au niveau du condenseur, or l'eau est une ressource rare au Burkina Faso (surtout dans les zones potentiellement éligibles à ces technologies). Cependant des solutions et des concepts innovants peuvent palier à ces problèmes. Les centrales à tour permettent d'atteindre les niveaux de température nécessaires à la réalisation d'un cycle de Brayton dans lequel l'air est utilisé comme fluide de travail. Le prix et la fiabilité du bloc de puissance pourrait être

amélioré via l'utilisation d'une turbine Tesla [2]. Enfin on peut également doter le champ solaire d'un système d'entraînement commun aux héliostats dans le but de réaliser une concentration performante et bon marché. C'est sur ce dernier aspect qu'est orienté ce mémoire avec pour objectif l'optimisation multi- échelles d'un champ solaire pour centrale à tour de petite puissance. Le terme multi-échelles signifie qu'il y a plusieurs échelles d'optimisation. On en dénombre trois : la taille des facettes composant les miroirs, la taille des héliostats, la disposition des héliostats dans le champ solaire. De multiples paramètres entrent en compte lors d'une telle optimisation. Un champ solaire est un système complexe composé de nombreux héliostats. Un grand nombre de calculs itératifs seront nécessaires pour déterminer la configuration optimale. Des logiciels sont en conséquence utilisés pour mener à terme ce travail. Etant donné qu'un système innovant tel qu'un champ solaire synchronisé nécessite une contrainte de disposition particulière (les héliostats doivent être alignés), la création d'un code adapté sera nécessaire. Les éléments qui composent une centrale à tour (champ solaire, récepteur, compresseur et turbine) sont interdépendants. Une présentation générale du projet constituera donc la première étape de notre travail.

1. Le concept de centrale à tour à air pressurisé

Le lecteur pourra trouver en annexe 1 quelques explications sur le fonctionnement d'une centrale thermique via un cycle de Brayton. Le fonctionnement d'une centrale solaire thermodynamique ne diffère de celui d'une centrale thermique standard que par la méthode utilisée pour apporter la chaleur au niveau de la chaudière. Si dans la deuxième on utilise des carburants fossiles, on obtient un résultat similaire en utilisant dans la première un rayonnement concentré. Il existe de multiples variantes pour concentrer le rayonnement solaire, mais on peut distinguer deux types de concentration : le resserrement ou la superposition du rayonnement. On utilise pour la première des surfaces réfléchissantes de formes géométriques variables, généralement paraboliques, qui requièrent une bonne précision de fabrication. La deuxième méthode consiste à superposer le rayonnement réfléchi via l'utilisation de miroirs plans. Elle permet essentiellement d'obtenir à la fois des concentrations élevées (en utilisant des structures de fabrication simple), une localisation fixe de la focale et une puissance importante au récepteur. C'est le type de concentration mise en œuvre dans une centrale à tour. La figure 2 illustre les deux variantes de concentration.

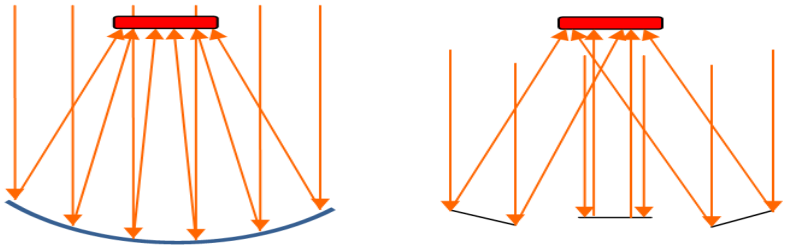


Figure 2 : illustration d'une concentration par resserrement, par superposition

Une centrale à tour comporte un champ solaire composé de miroirs qui s'orientent en permanence pour réfléchir le rayonnement qu'ils interceptent en un point fixe en haut de la tour. On appel ces miroirs des héliostats, ils sont eux mêmes composés de facettes disposées avec un léger jeu d'angle, ce qui permet de ramener la tache focale à la taille d'une facette au niveau du récepteur. On utilise donc dans une centrale à tour un double effet de superposition. Au sommet de la tour se trouve un récepteur (rôle de la chaudière) associé à un ensemble turbine-compresseur, dont le schéma de principe est donné en annexe 1.

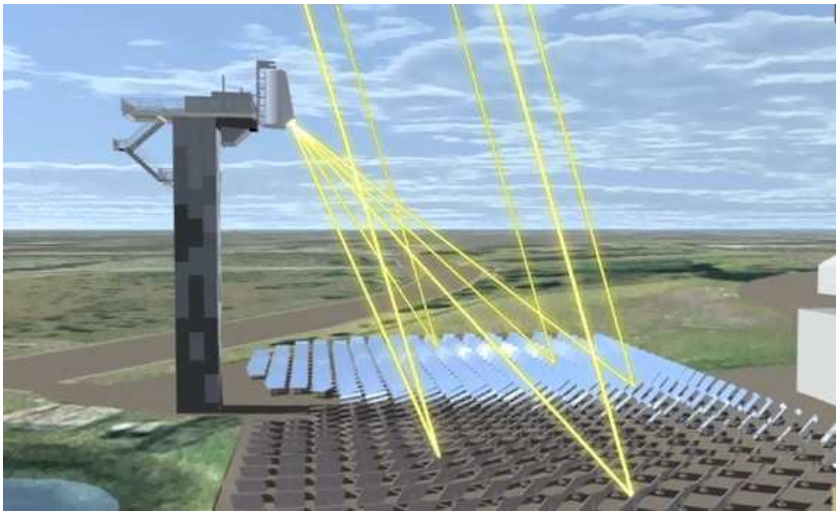


Figure 3: fonctionnement d'un champ solaire pour centrale à tour

2. Influence du récepteur sur la disposition du champ solaire

L'utilisation d'un cycle de Brayton nous amène à chauffer de l'air sous pression, or les propriétés convectives de l'air sont mauvaises. Convertir le rayonnement en chaleur à l'intérieur de la chambre de combustion via l'utilisation d'un récepteur volumétrique peut palier à ce problème. Toutefois fermer le récepteur avec un matériau transparent capable de supporter la différence de pression entre la chaudière et l'extérieur n'est pas une opération aisée. Diminuer la taille de la tache focale, c'est diminuer la surface de fermeture de la

chaudière. La capacité d'un matériau à résister à la pression variant de façon inverse à ses dimensions, le coût et la faisabilité du récepteur s'en trouvent améliorés. L'opération entraîne toutefois un coût supplémentaire au niveau du champ solaire mais celui-ci peut être surcompensé par les économies réalisées au récepteur. Voilà pourquoi un champ solaire ne peut être optimisé de manière isolée : un certain nombre de paramètres externes doivent donc être renseignés en tant qu'entrées pour le code. Dans le cas d'un récepteur surfacique, le problème est inversé : le flux solaire ne doit pas être trop concentré. En effet la chaleur étant transmise moins efficacement à l'air, il se crée des zones de surchauffe sur la face réceptrice. Ces surchauffes, dont la localité est aléatoire dans le temps, soumettent le métal à des variations de température importantes qui peuvent être brutales en cas de passage nuageux. Le récepteur est donc sujet à une fatigue thermique, des fissures apparaissent à court ou moyen terme sur la surface réceptrice qui doit alors être remplacée. On peut noter de plus, que les températures atteintes avec un récepteur surfacique sont inférieures à celle que l'on peut atteindre avec un récepteur volumétrique, ce qui pénalise le rendement. Quelque soit le choix du récepteur, la distribution en intensité du flux incident sur celui-ci est une donnée importante et un des objectifs de cette étude.

II. HYPOTHESES DE TRAVAIL

1. Les Facteurs de perte

Pour optimiser correctement un champ solaire il est nécessaire de recenser les facteurs de pertes spécifiques au champ d'héliostat. On recense actuellement cinq facteurs de pertes [3] qui sont décrit plus bas :

- Ombrages (générés par un obstacle entre le rayonnement incident et l'héliostat).

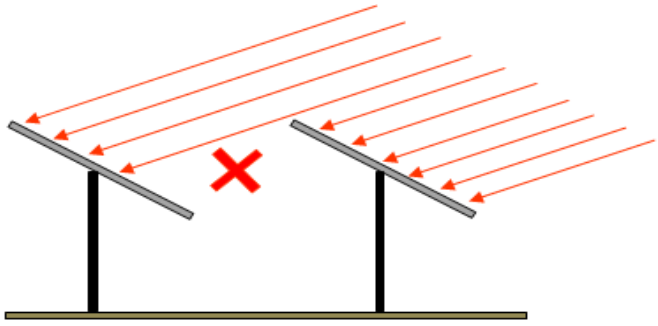


Figure 4: Représentation des effets d'ombrage

- Blocages (générés par un obstacle entre l'héliostat et le récepteur)

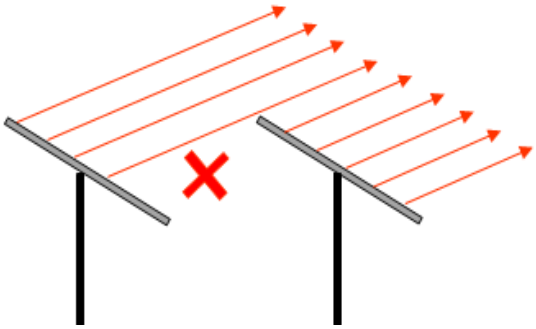


Figure 4: Représentation des effets de blocage

- Effet cosinus (généré par l'angle entre la normale à l'héliostat et le rayonnement incident)

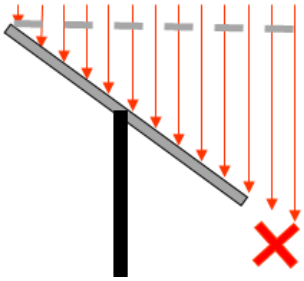


Figure 5: illustration de l'effet cosinus

- Coefficient de réflexion des héliostats (l'effet est ici très exagéré)

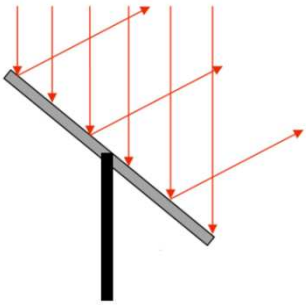


Figure 6 : représentation des pertes dues au coefficient de réflexion

- Les débordements au récepteur (générés par les erreurs de visée).

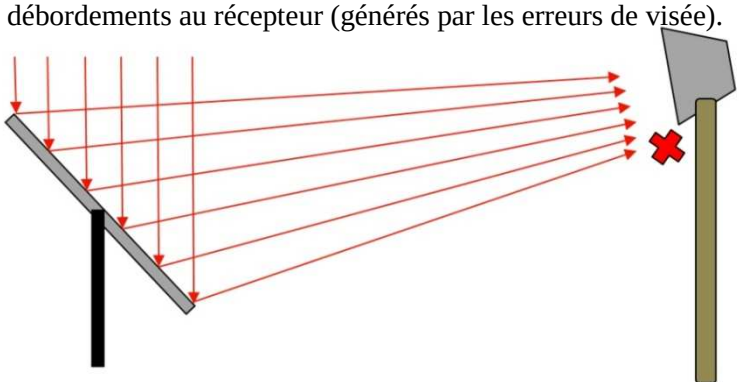


Figure 7 : illustration des débordements au récepteur

Tous ces effets sont résumés figure 9.

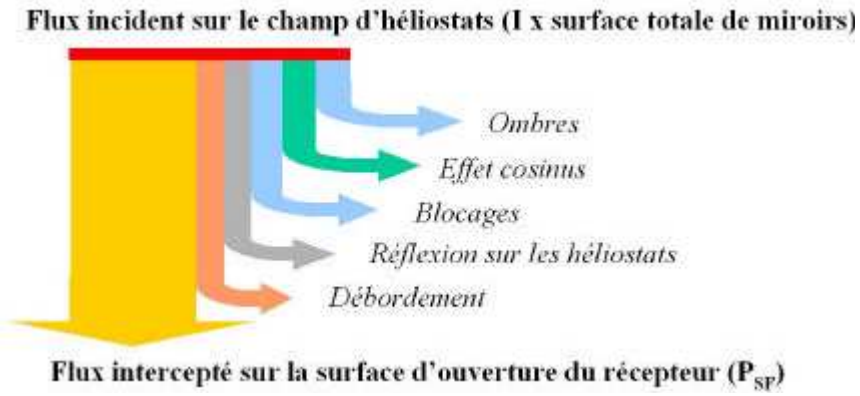


Figure 8: évolution du flux incident des héliostats au récepteur

En toute rigueur le rayonnement s'atténue lorsqu'il traverse le milieu atmosphérique (y compris sur le trajet séparant l'héliostat du récepteur), toutefois l'atténuation correspondant au parcours héliostat-récepteur est faible et pourra être négligée pour un champ de petite dimension.

2. Définition des objectifs

Si l'on souhaite concevoir un code, il faut au préalable être capable d'en définir le plus clairement possible les objectifs. Dans un premier temps, nous pourrions définir notre objectif comme suit : **obtenir une puissance maximale au récepteur, répartie sur une surface minimale pour un coût aussi faible que possible**. Cette façon d'envisager une optimisation est pourtant incomplète : d'autres facteurs doivent être pris en compte. Dans le cadre d'un tel projet, la faisabilité est un paramètre prioritaire. Il faudrait par exemple considérer la compacité de la disposition (la distance moyenne des héliostats au récepteur) : la précision de visée requise est d'autant plus faible que la distance moyenne des héliostats au récepteur est faible. De même le rendement et la longévité du bloc puissance sont fortement compromis par les variations de la ressource solaire au niveau du récepteur : une disposition capable de lisser ou du moins de ne pas trop amplifier ces variations serait appréciable. Enfin le ratio **taille de la focale/prix du champ** est difficile à déterminer. En effet à l'état d'avancement du projet, il nous est impossible d'évaluer précisément le prix du récepteur. Dès lors, il serait illusoire de quantifier les gains générés par une réduction en taille de la focale. Optimiser un champ solaire pour centrale à tour est donc une démarche complexe qui nécessite de prendre en compte à la fois des paramètres quantitatifs et des paramètres qualitatifs. En conséquence

notre choix s'est porté sur la conception d'un code d'optimisation-simulation. Ce code devra comporter toutes les fonctionnalités qui permettront à l'utilisateur d'apprécier la qualité de la disposition. En plus des facteurs de pertes (effet cosinus, ombrages, blocages), il devra pouvoir fournir des données telles l'évolution de la puissance au récepteur où la distance moyenne des héliostats au récepteur. Une donnée particulièrement appréciable serait la répartition de la puissance au récepteur 'en temps réel', qui n'est actuellement fournie par aucun logiciel (voir le fonctionnement de Soltrace p.15-17). Il sera donc laissé à l'utilisateur le soin (et la possibilité) d'apprécier de multiples critères qualitatifs qui ne pourraient pas être traités par un algorithme. L'utilisateur élaborera ainsi des scénarii qui permettront au responsable de projet de choisir ce qu'il estime être le meilleur compromis.

3. Définition des paramètres d'influence

Pour organiser la conception du code, nous allons dans un premier temps lister tous les paramètres susceptibles d'influer sur les performances du champ solaire (et donc sur la disposition générée). Ces paramètres représenteront les entrées du code. De la même manière, nous listons tous les facteurs de pertes influencés par ces paramètres, ils représenteront les sorties du code qui pourront être exprimées soit par des représentations graphiques, soit par des représentations numériques.

Nous trouvons au niveau des paramètres d'influence:

- La hauteur de tour
- L'inclinaison du récepteur
- L'intensité du rayonnement et son orientation (la date précise de l'optimisation)
- Les dimensions des héliostats et des facettes
- La précision de visée des héliostats
- Le coefficient de réflexion des héliostats

Et au niveau des facteurs de perte :

- L'effet cosinus
- Les ombrages
- Les blocages
- Les débordements au récepteur
- Les pertes par réflexion
- L'atténuation atmosphérique du rayonnement

4. Chronologie de la conception

Le travail s'organisera comme suit:

- Détermination de l'objectif du code
- Détermination des facteurs de pertes qui devront être pris en compte par le code
- Quantification des facteurs de perte
- Programmation et vérification d'un code capable de calculer et d'afficher les effets listés ci-dessous (qui sont dans une très large mesure les facteurs de pertes majoritaires associés à un champ solaire pour centrale à tour).
  - Effet cosinus ≫ Validation Windelsol
  - Ombrages ≫ Validation Windelsol
  - Blocages ≫ Validation Windelsol

La multiplication de l'intensité par ces facteurs, par le coefficient de réflexion des miroirs ainsi que par la surface d'héliostats renseigne la puissance disponible au récepteur.

- Détermination d'un algorithme pour optimiser les dispositions.

III. MATERIEL ET METHODE

De nombreux logiciels sont actuellement destinés à l'optimisation des centrales à tour. Concernant l'optimisation du champ solaire, on distingue deux grandes familles :

- Les logiciels utilisés pour l'optimisation de la géométrie du champ solaire : Windelsol, HFLCAL, Rcell, HGM.
- Les logiciels permettant une analyse plus fine du système optique : Soltrace, Mirval, Tonatiuh, Raytrace3D … Ces derniers sont la plupart du temps utilisés pour établir la répartition de la puissance au niveau du récepteur une fois le champ solaire optimisé. Ils sont également adaptés pour déterminer la forme du concentrateur secondaire utilisé au niveau du récepteur dans le cas des centrales à air pressurisé.

De tous les logiciels précédemment cités, Windelsol et Soltrace sont les plus utilisés. Certaines de leurs fonctionnalités ont fortement influencé la conception du code et feront donc l'objet d'une présentation commentée. Le principe de fonctionnement d'un nouveau code nommé HGM [4] (Héliostat Growth Method), développé par CIEMAT dont l'environnement de travail est MATLAB est présenté en annexe 2. Un tableau récapitulatif

décrivant les principaux logiciels d'optimisation est donné en annexe 3.

1. Description de Windelsol

1.1 Présentation

Windelsol est un logiciel nouvelle génération développé conjointement par AICIA, CIEMAT et SOLUCAR en 2001 [5]. La possibilité de travailler sous un environnement Windows le rend plus familier à l'utilisation que ses prédécesseurs. Ce logiciel dispose de nombreuses fonctionnalités et permet d'afficher les matrices d'efficacité associées à chaque facteur de perte dont on trouvera une illustration à la figure 10 [6] qui est analysée plus en détail ci-dessous.

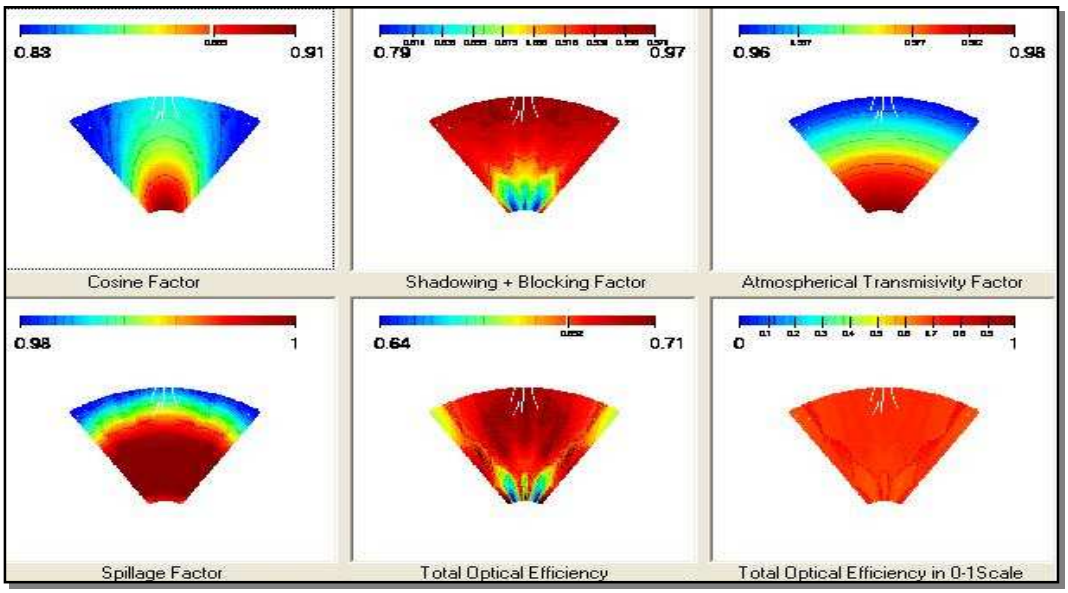


Figure 10: matrices d'efficacité générées par Windelsol

2.1 Commentaires

Nous pouvons tirer de cette image beaucoup d'enseignements utiles. Nous remarquons en premier lieu que l'utilisation des couleurs dans les matrices d'efficacité est particulièrement appréciable : un coup d'œil sur la représentation du champ permet de prendre immédiatement connaissance de l'ampleur des différents effets. Les matrices d'efficacité sont donc un outil de travail que nous choisissons d'intégrer dans notre propre logiciel. Nous pouvons également comparer en ampleur les différents effets : l'effet cosinus est l'effet majoritaire, suivi du cumul des ombrages et des blocages : ils devront impérativement être pris en compte dans notre code. L'atténuation atmosphérique ne génère pas de pertes supérieures à 4% de la puissance réfléchie. Il faut ajouter à cela que les dimensions des dispositions pour l'instant calculées avec Windelsol sont de l'ordre du kilomètre alors que

celles du projet de centrale à tour au 2ie n'excèderont probablement pas 50 mètres. Les pertes engendrées par l'atténuation atmosphérique pourront donc être considérées comme négligeables, ou du moins suffisamment faibles pour ne pas influer significativement sur la disposition du champ. Les débordements au récepteur sont les effets minoritaires : ils n'excèdent pas 2%. On peut remarquer de plus qu'étant donné le faible nombre de centrales à tour réalisées, il n'existe pas encore de récepteurs à air pressurisé commercialisés. Dans un tel projet, chaque composant doit être conçu ou adapté pour disposer des fonctions désirées. Dès lors il est plus prudent de programmer la conception du récepteur après la mise en fonctionnement du champ solaire : on évite ainsi tout effet de débordement. Enfin puisqu'il n'a pas été possible de nous procurer le logiciel Windelsol dans les délais de cette étude, nous utiliserons cette image pour comparer (au niveau de la forme) les résultats de notre logiciel avec ceux de Windelsol et ainsi valider ou non les mises en équation et les éventuelles approximations.

3.1 Fonctionnement

L'utilisation de Windelsol peut se faire suivant deux cas de figure :

- Génération d'une disposition optimale
- Calcul du comportement d'un champ d'héliostat défini

Dans la deuxième configuration le code offre la possibilité de calculer la performance optique soit pour un instant déterminé, soit sur une base annuelle. Le système spécifié dans cette demande peut être le système optimal calculé par le code.

On donne, figure 11, la représentation de la performance globale du champ d'héliostat.

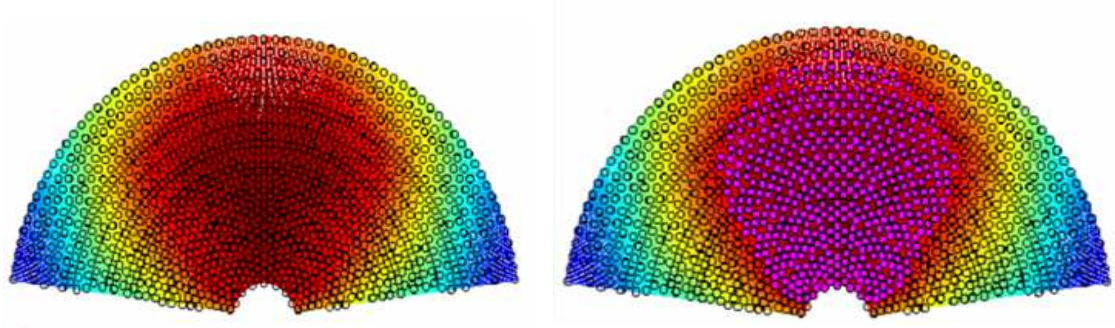


Figure 9: Représentation sous Windelsol de la zone d'efficacité du champ solaire, de la disposition optimale.

Cette figure nous permet de prendre connaissance du type de disposition générée par Windelsol. La disposition influe sur les matrices d'efficacité, si l'on souhaite comparer celles générées par Windelsol avec nos propres matrices, il faut utiliser une disposition de forme similaire. On voit sur la figure 10 que la disposition (les héliostats sont représentés par des

points violets) est de forme circulaire quinconce, nous programmerons une telle disposition à titre comparatif.

2. Description de Soltrace

Soltrace est un logiciel permettant une analyse fine de systèmes optiques, il est généralement utilisé en complément de Windelsol pour déterminer la répartition de la puissance au niveau du récepteur. Cette donnée est en effet indispensable pour calculer l'inclinaison optimale du récepteur, concevoir le concentrateur secondaire (s'il y en a un) et le récepteur lui-même. Elle peut également être utile dans le choix du nombre de facettes par héliostat. La méthode utilisée est le lancer de rayon, une simulation s'effectue en trois étapes :

- Définition de la source lumineuse (intensité, orientation)
- Définition du système optique (coordonnées, forme, orientation)
- Génération de rayons et décompte

3. Technique du lancer de rayon

Soltrace génère des rayons, applique les lois de l'optique géométrique et détermine la direction de leur réflexion. Il est également capable de comptabiliser les rayons interceptés par un élément définit au préalable.

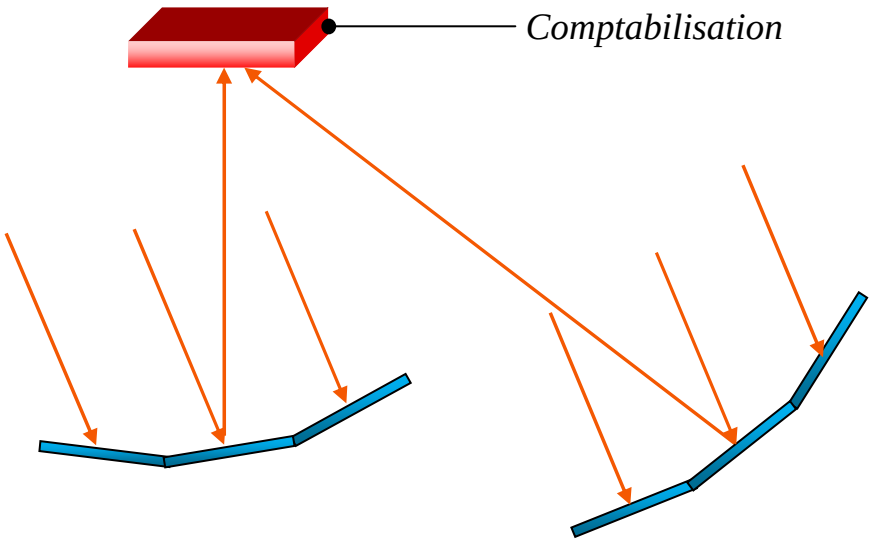


Figure 12 : illustration de la méthode du lancer de rayons

L'utilisateur peut choisir un élément et y générer un certain nombre de rayons. Les rayons sont créés suivant l'angle préalablement entré dans la définition du rayonnement et de manière aléatoire au travers l'ouverture de l'élément, c'est pourquoi il est conseillé d'en

utiliser un nombre important (au moins dix mille par élément). A l'intersection de chaque rayon et de l'élément, Soltrace calcule la trajectoire du rayon réfléchi puis la prolonge jusqu'à la prochaine intersection. Dès qu'un rayon croise un élément il est comptabilisé.

On peut donc disposer le système de la manière suivante :

Le champ solaire est composé d'éléments qui représentent les héliostats, un élément fait office de récepteur. Le logiciel génère un nombre important de rayons sur chaque élément du champ puis calcule la trajectoire des rayons réfléchis et comptabilise ceux interceptés par le récepteur (voir figure 12). La finalité consiste à générer puis à afficher le profil de puissance reçue au niveau du récepteur (voir figure 13).

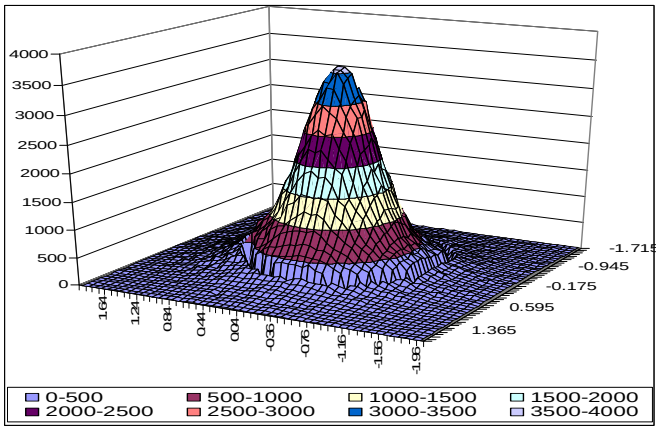


Figure 13 : Profil d'intensité au récepteur

4. Commentaires

Bien qu'il soit capable de fournir des résultats très fins, Soltrace ne souffre pas moins de deux défauts majeurs :

- Le temps de calcul : plusieurs heures pour une disposition
- Soltrace est un logiciel statique

Le temps de calcul relativement long s'explique par le fait que Soltrace génère des rayons de manière aléatoire sur les éléments à étudier. Un grand nombre de rayon est donc nécessaire pour couvrir de manière homogène la surface à étudier. Dans le cas de l'étude de surfaces planes ou quasi-planes (comme celle des héliostats d'une centrale à tour), cette génération aléatoire n'a plus d'intérêt mais se révèle même être un handicap. Une génération 'organisée' serait tout aussi efficace et diminuerait significativement les temps de calculs. Le deuxième défaut de Soltrace tient au fait qu'il s'agit d'un logiciel statique, c'est-à-dire qu'il ne permet de faire une simulation qu'à un instant donné. Si l'on souhaite par exemple obtenir la répartition de puissance pour un autre instant, il faut recalculer puis modifier une à une les orientations de tous les héliostats du champ solaire. Cette contrainte limite fortement le nombre de

répartitions générées et donc nuit à la qualité de l'optimisation.

Puisqu'à termes nous disposerons d'un logiciel capable de calculer en temps réel l'orientation des héliostats (condition indispensable au calcul des facteurs de perte), nous pourrions lui adjoindre en plus la 'fonctionnalité Soltrace'. Ainsi générer une nouvelle répartition du flux solaire au récepteur ne prendrait que quelques minutes, sans nécessiter à chaque essai la reconfiguration intégrale du champ solaire. Cette fonctionnalité permettrait de créer le premier logiciel 'complet' d'optimisation des champs solaires pour centrale à tour.

5. Choix du logiciel de programmation

Le choix du langage de programmation est une étape importante dans la conception du code. Il existe dans le monde de l'open source toute une série de langages gratuits parmi lesquels le C/C++ s'impose sans conteste comme une référence absolue à laquelle tout informaticien sérieux doit se frotter tôt ou tard. Il est malheureusement rébarbatif et compliqué, trop proche de la machine. Sa syntaxe est peu lisible et contraignante. La mise au point d'un gros logiciel écrit en C/C++ est longue et pénible. (Les mêmes remarques valent aussi dans une large mesure pour le langage *Java*). D'autre part, la pratique moderne de ce langage fait abondamment appel à des générateurs d'applications et autres outils d'assistance très élaborés tels *C++Builder*, *Kdevelop*, etc. Ces environnements de programmation peuvent certainement se révéler très efficaces entre les mains de programmeurs expérimentés, mais ils proposent d'emblée beaucoup trop d'outils complexes, et ils présupposent de la part de l'utilisateur une maîtrise totale du logiciel. L'idéal serait certainement d'apprendre à les utiliser mais il faut cependant bien admettre que le temps dont nous disposons est limité. Pour nos besoins, il semble préférable d'utiliser un langage de plus haut niveau, moins contraignant, à la syntaxe plus lisible. Notre choix s'est porté sur le langage Python, développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles. Ce langage de programmation présente en effet de nombreux avantages :

- Python est **portable**, non seulement sur les différentes variantes d'*Unix*, mais aussi sur les OS propriétaires: *MacOS*, *BeOS*, *NeXTStep*, *MS-DOS* et les différentes variantes de *Windows*.
- Python est **gratuit**, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des **projets complexes** de plusieurs dizaines de milliers de lignes.

- La **syntaxe de Python est très simple** et, combinée à des **types de données évoluées** (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C/C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue [5].
- Python est dynamiquement typé, c'est-à-dire que tout objet manipulable par le programmeur possède un type bien défini à l'exécution, mais qui n'a pas besoin d'être déclaré à l'avance (contrairement à C/C++).

Toutes ces raisons font que python s'est imposé comme le choix le plus judicieux

IV. CONCEPTION DU CODE

1. Calcul de l'orientation du rayonnement incident

Le code comportera comme entrée l'heure précise et la date à laquelle il devra optimiser la disposition du champ solaire. Il devra être capable de déterminer l'orientation du rayonnement à partir de ces données.

Nous donnons ci-dessous les formules de calcul de position du soleil.

On calcul d'abord le numéro du jour de l'année, donné par la formule suivante :

$$n = \frac{285 \times m}{9} - 2 \times \frac{m+9}{12} + j - 30 \quad (1)$$

La deuxième équation renseigne l'horaire local.

$$h_{loc} = h + \frac{\min}{60} + \frac{\sec}{3600} \quad (2)$$

Le temps solaire moyen intègre une correction liée à la longitude du lieu (il faut corriger l'écart induit par la différence entre la longitude du lieu concerné et la longitude du méridien sur lequel se base le fuseau horaire du lieu).

$$TSM = h_{loc} + \frac{l}{15} \quad (3)$$

La déclinaison (l'angle que fait l'axe de rotation de la terre avec la normale au plan de

l'écliptique) affecte l'heure où la course du soleil croise la longitude du lieu en question. On corrige ce décalage par l'équation du temps dont la formule (donnée plus bas) prend en compte la position absolue de la terre autour du soleil (soit indirectement le numéro du jour de l'année).

$$\delta = \arcsin\left(0{,}39795 \times \cos\left(0{,}98563 \times (n - 173)\right)\right) \quad (4)$$

La position absolue de la terre autour du soleil est reliée de manière simple au numéro du jour de l'année puisque la terre effectue une rotation en 365,242 jours.

$$p_{abs} = 360 \times \frac{n-1}{365{,}242} \quad (5)$$

$$Eot = 0{,}258 \times \cos(p_{abs}) - 7{,}416 \times \sin(p_{abs}) - 3{,}648 \times \cos(2p_{abs}) - 9{,}228 \times \sin(2p_{abs}) \quad (6)$$

Le temps solaire vrai est donc le temps solaire moyen corrigé par l'équation du temps.

$$TSV = TSM + \frac{Eot}{60} \quad (7)$$

L'angle horaire w est le déplacement angulaire du soleil à l'Est ou à l'Ouest du méridien local du à la rotation de la terre (15° par heure).

$$w = (TSV - 12) \times 15 \quad (8)$$

On donne enfin les deux formules de calcul associées à l'élévation et à l'azimut solaire.

$$h_{sol} = \arcsin(\sin\delta \times \sin L + \cos\delta \times \cos w \times \cos L) \quad (9)$$

$$a_{sol} = -180 + \arccos\left(\sin\delta \times \cos L - \frac{\cos\delta \times \cos w \times \sin L}{\cos h_{sol}}\right) \quad (10)$$

2. Calcul de l'orientation de l'héliostat

L'orientation de la normale à l'héliostat est une donnée indispensable au calcul de l'effet cosinus, des ombrages et des blocages. Le calcul de cette orientation est donc donné ci-dessous. Pour déterminer l'orientation de la normale, nous nous basons sur les lois de l'optique géométrique : la normale à l'héliostat est la bissectrice entre le rayon incident et le rayon réfléchi, contenue dans le même plan (comme illustré figure 14).

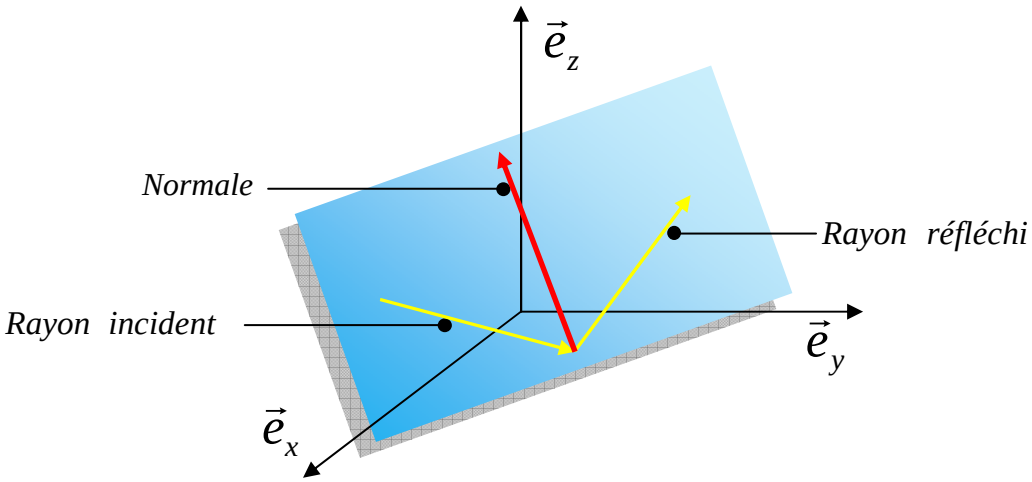


Figure 14: représentation du rayon incident, réfléchi et de la normale à l'héliostat

Pour éviter de passer par des calculs trop lourds, nous construirons la normale comme la somme de deux vecteurs. En effet si l'on additionne deux vecteurs non colinéaires de même norme et d'orientations opposées, on obtient un vecteur à la fois contenu dans le même plan et bissecteur des deux vecteurs origines (dessin de gauche de la figure 15). D'autre part nous placerons un repère fixe dont l'origine est confondue avec l'axe de rotation de l'héliostat. Les coordonnées du récepteur représentent alors un vecteur de même orientation que le rayonnement réfléchi (dessin de droite de la figure 15).

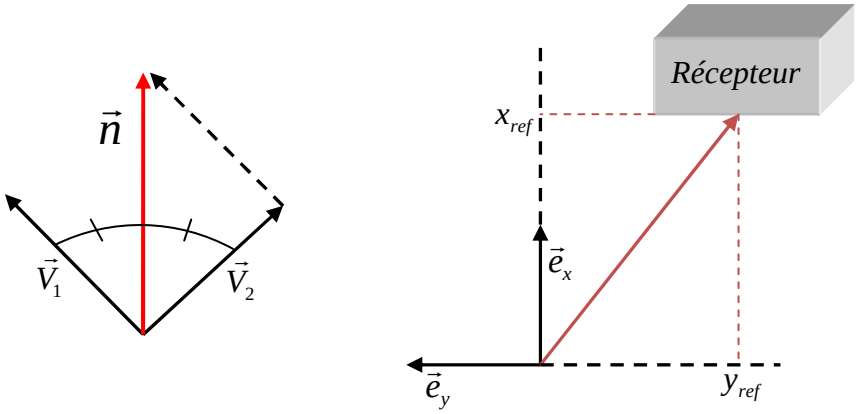


Figure 15: illustration de la méthode utilisée, choix du repère

Nous disposons des coordonnées angulaires du rayon incident, et des coordonnées cartésiennes du rayon réfléchi, nous allons retranscrire ces deux vecteurs en vecteurs unitaires, il ne nous restera ensuite qu'à les sommer.

3. Vecteur réfléchi

Si nous appelons $X_{rec}, Y_{rec}, Z_{rec}$ les coordonnées du récepteur, on remarque que ces coordonnées définissent un vecteur de même orientation que le vecteur réfléchi, il est simplement « trop grand ». Il suffit donc de le diviser par sa norme pour déterminer

$X_{ref}, Y_{ref}, Z_{ref}$ qui définit le vecteur unitaire du rayon réfléchi.

$$X_{ref} = \frac{X_{rec}}{\sqrt{X_{rec}^2 + Y_{rec}^2 + Z_{rec}^2}} \quad (11)$$

$$Y_{ref} = \frac{Y_{rec}}{\sqrt{X_{rec}^2 + Y_{rec}^2 + Z_{rec}^2}} \quad (12)$$

$$Z_{ref} = \frac{Z_{rec}}{\sqrt{X_{rec}^2 + Y_{rec}^2 + Z_{rec}^2}} \quad (13)$$

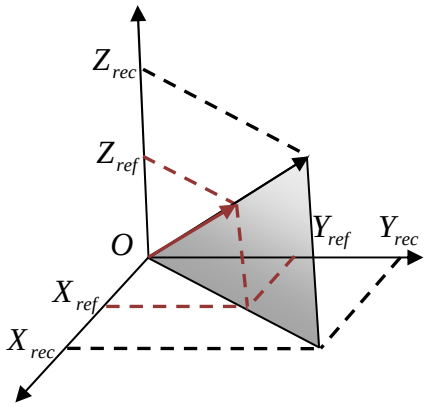


Figure 16: représentation du vecteur réfléchi et du vecteur réfléchi normé

4. Vecteur incident

Si $X_{inc}, Y_{inc}, Z_{inc}$ définissent le vecteur incident unitaire. Il est facile de voir que :

$$\tan h = \frac{Z_{inc}}{\sqrt{X_{inc}^2 + Y_{inc}^2}} \quad (14)$$

$$\tan a = -\frac{Y_{inc}}{X_{inc}} \quad (15)$$

$$X_{inc}^2 + Y_{inc}^2 + Z_{inc}^2 = 1 \quad (16)$$

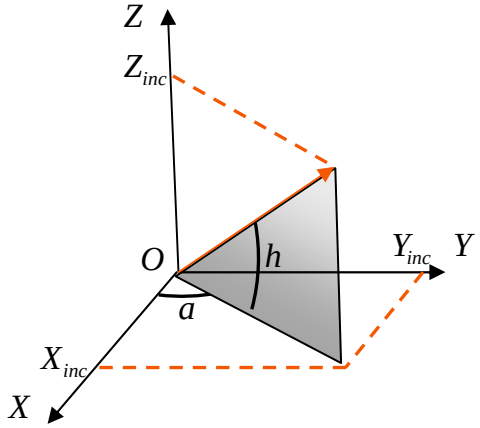


Figure 17: représentation du vecteur incident

En injectant (14) et (15) dans (16) nous avons :

$$X_{inc}^2 + X_{inc}^2 \tan^2 a + X_{inc}^2 \tan^2 h + X_{inc}^2 \tan^2 a \tan^2 h = 1$$

Qui donne :

$$X_{inc} = \frac{1}{\sqrt{1 + \tan^2 a + \tan^2 h + \tan^2 a \tan^2 h}} \quad (17)$$

$$Y_{inc} = \frac{-\tan a}{\sqrt{1 + \tan^2 a + \tan^2 h + \tan^2 a \tan^2 h}} \quad (18)$$

$$Z_{inc} = \tan h \sqrt{\frac{1 + \tan^2 a}{1 + \tan^2 a + \tan^2 h + \tan^2 a \tan^2 h}} \quad (19)$$

Nous obtenons ainsi l'expression du vecteur normal à l'héliostat :

$$X_n = \frac{1}{\sqrt{1+\tan^2 a+\tan^2 h+\tan^2 a\,\tan^2 h}} + \frac{X_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \quad (20)$$

$$Y_n = \frac{-\tan a}{\sqrt{1+\tan^2 a+\tan^2 h+\tan^2 a\,\tan^2 h}} + \frac{Y_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \quad (21)$$

$$Z_n = \tan h\sqrt{\frac{1+\tan^2 a}{1+\tan^2 a+\tan^2 h+\tan^2 a\,\tan^2 h}} + \frac{Z_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \quad (22)$$

$$Soit\ \vec{n} = \begin{pmatrix} \frac{1}{\sqrt{1+tan^2 a+tan^2 h+tan^2 a\,tan^2 h}} + \frac{X_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \\ \frac{-tan a}{\sqrt{1+tan^2 a+tan^2 h+tan^2 a\,tan^2 h}} + \frac{Y_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \\ tan\ h\sqrt{\frac{1+tan^2 a}{1+tan^2 a+tan^2 h+tan^2 a\,tan^2 h}} + \frac{Z_{rec}}{\sqrt{X_{rec}^2+Y_{rec}^2+Z_{rec}^2}} \end{pmatrix}$$

5. Quantification de l'effet cosinus

Une fois déterminée la normale à l'héliostat, l'effet cosinus peut être calculé de manière simple. Il s'agit en effet de calculer le cosinus de l'angle que fait la normale avec le rayonnement incident. Nous connaissons à présent les coordonnées du vecteur incident et de la normale à l'héliostat, nous obtiendrons le cosinus en effectuant le produit scalaire de ces deux vecteurs :

$$\begin{pmatrix} X_{inc} \\ Y_{inc} \\ Z_{inc} \end{pmatrix} \bullet \begin{pmatrix} X_n \\ Y_n \\ Z_n \end{pmatrix} = X_{inc}X_n + Y_{inc}Y_n + Z_{inc}Z_n = \|V_{inc}\| \times \|V_n\| \times \cos\theta \quad (23)$$

$$Puisque\ \|V_{inc}\| = 1,\ il\ vient:$$

$$\cos\theta = \frac{X_{inc}X_n + Y_{inc}Y_n + Z_{inc}Z_n}{\sqrt{X_n^2+Y_n^2+Z_n^2}} \quad (24)$$

Ce qui quantifie l'effet cosinus.

6. Quantification de l'effet d'ombrage

6.1. Détermination de la méthode

La quantification des effets d'ombrage ou de blocage est une entreprise complexe, le logiciel HGM, développé par Ciemat, utilise d'ailleurs des approximations (non décrites) pour le calcul de ces effets. La littérature technique sur les logiciels d'optimisation des champs

solaires étant particulièrement pauvre, nous n'avons aucune information sur les techniques de mise en équation ou les approximations utilisées par les autres logiciels. En conséquence il nous faut développer nos propres techniques et approximations en considérant les particularités du projet en question. Vis-à-vis du champ solaire, le concept qui doit être testé est la mise en commun du système d'entraînement des héliostats. Les résultats attendus sont une diminution du coût du système de suivi, un allègement de la structure des héliostats et une amélioration en qualité de la focale. Les deux derniers avantages découlent de l'utilisation de petits héliostats, elle-même rendue possible par la mise en commun de l'entraînement (et donc la réduction du nombre de moteurs et systèmes de commande utilisés). Quoiqu'il en soit l'utilisation de petits héliostats nous permet une approximation qui va considérablement simplifier la mise en équation des ombrages et des blocages. Nous considérerons dans la suite que deux héliostats voisins sont parallèles. Cette approximation n'est valable que lorsque les dimensions de l'héliostat sont faibles devant les dimensions du champ solaire. En effet si cette dernière hypothèse est vérifiée les rayons réfléchis de deux héliostats voisins sont quasiment parallèles (les héliostats voient le récepteur sous un angle très proche). D'autre part ils reçoivent le même rayonnement incident. En conséquence puisque la normale est la bissectrice de l'incident et du réfléchi nous pouvons conclure que les deux normales sont quasiment parallèles : les héliostats ont presque la même orientation.

Cette approximation nous permet entre autres de considérer les limites des ombrages ou des blocages parallèles aux côtés de l'héliostat où l'on évalue l'effet. Elle ne simplifie pas seulement le calcul des parties de l'héliostat ombragées où bloquées mais surtout le calcul des zones d'intersections lorsque plusieurs obstacles génèrent des ombrages et/ou des blocages qui se superposent sur un seul héliostat. Le calcul des ombrages s'effectuera en deux étapes : dans un premier temps il faut déterminer l'emplacement des obstacles à l'origine de l'effet et leur position relative à l'héliostat sur lequel on évalue l'effet. Il nous suffira ensuite d'une seule information pour évaluer l'ombrage : les coordonnées de l'intersection du projeté de l'axe de rotation de l'obstacle sur le plan de l'héliostat ombragé suivant la direction du rayonnement incident (la méthode sera détaillée plus loin). En vue d'illustrer la méthode, nous prendrons l'exemple d'une disposition simple pour déterminer les coordonnées des obstacles générant les effets d'ombrage : une disposition alignée à la fois Est-Ouest et Nord-Sud (voir figure 18). La distance entre deux voisins sera notée a, elle est la même suivant les deux axes.

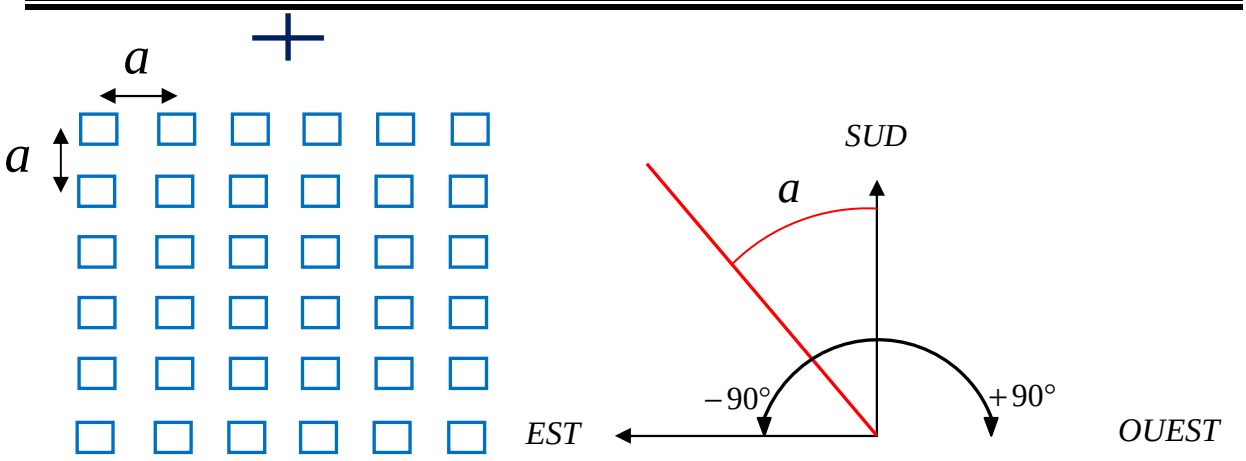


Figure 18 : disposition type alignée, comptabilisation de l'azimut dans le repère fixe associé à l'héliostat

L'effet d'ombrage a pour origine l'interaction entre le rayonnement incident et un ou plusieurs héliostats situés plus en avant (dans la direction de l'azimut solaire). Il faut donc prendre en compte l'azimut solaire pour déterminer les coordonnées des obstacles susceptibles de générer l'ombrage. Nous avions associé à chaque héliostat un repère fixe situé au niveau de son axe de rotation (p.22), ce repère ainsi que la comptabilisation de l'azimut sont illustrés figure 18. Dans le cas de notre disposition et en notant x_{omb} , y_{omb} les coordonnées des obstacles potentiels, nous pouvons distinguer les cas suivants :

- $-90° < azimut \leq -45°$: $x_{omb1} = 0$, $y_{omb1} = a$; $x_{omb2} = a$, $y_{omb2} = a$
- $-45° < azimut \leq 0°$: $x_{omb1} = a$, $y_{omb1} = a$; $x_{omb2} = a$, $y_{omb2} = 0$
- $0° < azimut \leq 45°$: $x_{omb1} = a$, $y_{omb1} = 0$; $x_{omb2} = a$, $y_{omb2} = -a$
- $45° < azimut \leq 90°$: $x_{omb1} = a$, $y_{omb1} = -a$; $x_{omb2} = 0$, $y_{omb2} = -a$

Munis des coordonnées des obstacles relatives à l'héliostat sur lequel l'effet doit être évalué, nous allons procéder de la manière suivante :

Dans un premier temps l'orientation de l'héliostat ombragé est calculée suivant la méthode exposée p.21-23. Cette orientation nous permet de déterminer l'équation du plan de l'héliostat dans le repère fixe qui lui est associé. Nous déterminons ensuite, toujours dans le même repère, l'équation de la droite passant par l'axe de rotation de l'obstacle et de même orientation que le rayonnement incident. Les coordonnées de l'intersection entre le plan de l'héliostat et la droite constituent toutes les informations nécessaires au calcul de l'effet d'ombrage. En effet, d'après l'approximation considérée, les orientations de l'héliostat ombragé et de ses plus proches voisins (les obstacles) sont considérées identiques. Dès lors les limites des ombrages générés sont parallèles aux côtés de l'héliostat ombragé. Si nous associons à l'héliostat ombragé un repère tournant, et qu'il nous est possible de déterminer les

coordonnées de l'intersection dans ce repère, alors le calcul de la portion recouverte par l'ombre est élémentaire. Il suffit en effet d'ajouter ou de retirer le demi côté de l'héliostat aux coordonnées de l'intersection pour déterminer les limites de l'ombrage. La figure 19 résume la méthode.

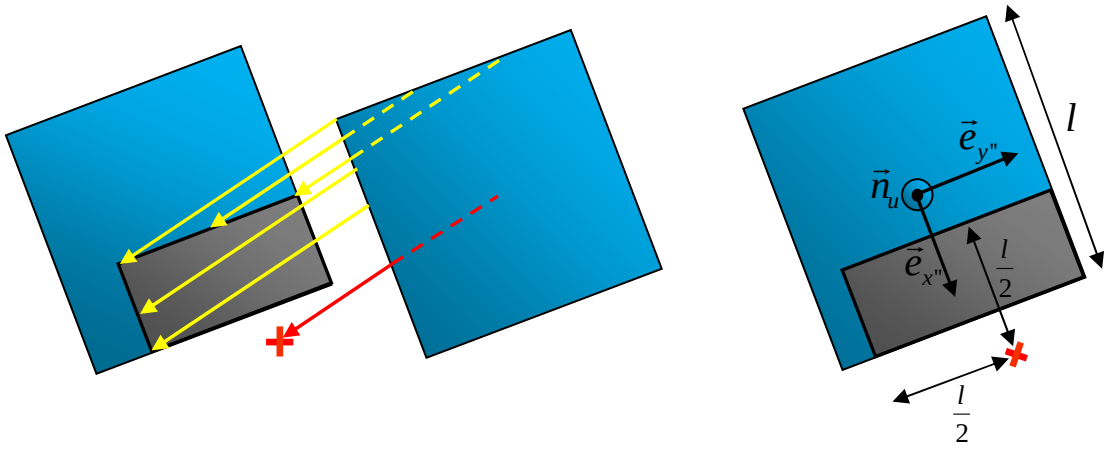


Figure 19 : méthode utilisée pour le calcul des effets d'ombrage

Nous donnons à titre d'exemple la formule permettant de calculer l'ombrage dans le cas simple exposé figure 19. En considérant $x_{\text{int}er}$ et $y_{\text{int}er}$ les coordonnées de l'intersection dans le repère tournant associé à l'héliostat et s_{omb} la surface ombragée de l'héliostat, nous avons :

$$s_{omb} = (l - x_{\text{int}er}) \times (l - y_{\text{int}er}) \quad (25)$$

D'où la formule exprimant l'efficacité en rapport aux ombrages :

$$e_{omb} = 1 - \frac{(l - x_{\text{int}er}) \times (l - y_{\text{int}er})}{l^2} \quad (26)$$

6.2 Mise en équation

6.2.1. Changement de repère

Nous utilisons pour le calcul des effets d'ombrage deux repères associés à l'héliostat dont l'origine est confondue avec l'axe de rotation de ce dernier. Il faut en conséquence établir les formules de changement de repère entre le repère fixe et le repère en rotation. Nous connaissons déjà dans le repère fixe l'expression cartésienne du vecteur normal à l'héliostat. Normer ce vecteur à l'unité est une opération simple, nous obtenons :

$$\vec{n}_u = \begin{pmatrix} \dfrac{X_n}{\sqrt{X_n^2 + Y_n^2 + Z_n^2}} \\ \dfrac{Y_n}{\sqrt{X_n^2 + Y_n^2 + Z_n^2}} \\ \dfrac{Z_n}{\sqrt{X_n^2 + Y_n^2 + Z_n^2}} \end{pmatrix}$$

Où $\vec{n}_u$ est le vecteur normal unitaire. Nous attachons à l'héliostat un repère tournant : $\vec{n}_u$ (le vecteur normal unitaire de l'héliostat) sera le vecteur $\vec{e}_{z''}$ de ce nouveau repère, on trouvera une représentation figure 20.

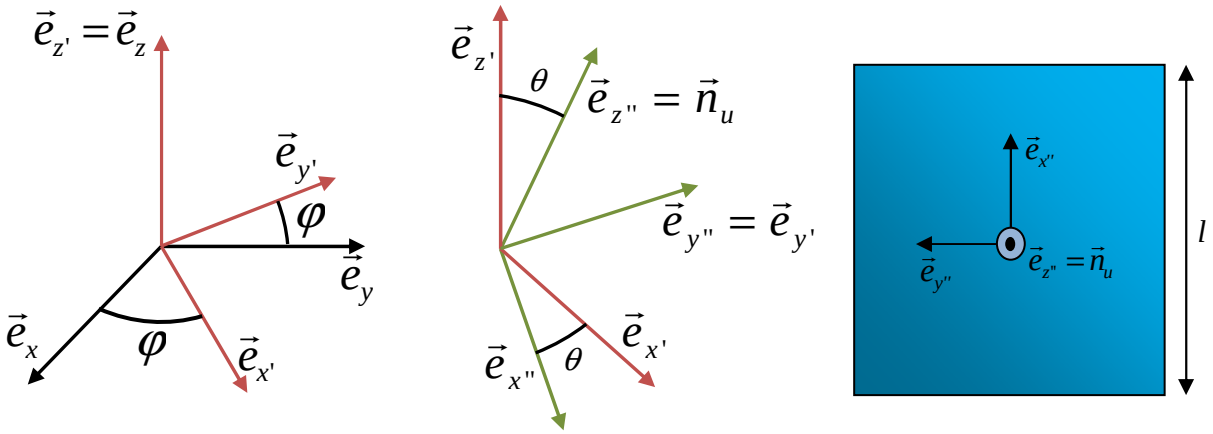


Figure 20 : illustration du changement de repère, disposition du repère tournant sur l'héliostat

Naturellement dans le repère tournant tous les points éléments de l'héliostat ont des coordonnées fixes.

Ce repère peut être obtenu à partir du repère fixe par deux rotations successives (illustrées figure 20) : une rotation d'angle φ autour de $\vec{e}_z$ nous amène à un repère intermédiaire définit par $\vec{e}_{x'}, \vec{e}_{y'}\ \vec{e}_{z'}$. A partir de ce repère intermédiaire, on peut obtenir l'équation du nouveau repère par une rotation d'angle θ autour de $\vec{e}_{y'}$.

Pour le repère intermédiaire, nous avons :

$\vec{e}_{z'} = \vec{e}_z$; $\vec{e}_{x'} = cos\,\varphi\,\vec{e}_x + sin\,\varphi\,\vec{e}_y$; $\vec{e}_{y'} = -sin\,\varphi\,\vec{e}_x + cos\,\varphi\,\vec{e}_y$

Pour la deuxième rotation :

$\vec{e}_{y''} = \vec{e}_{y'}$; $\vec{e}_{z''} = sin\,\theta\,\vec{e}_{x'} + cos\,\theta\,\vec{e}_{z'}$; $\vec{e}_{x''} = cos\,\theta\,\vec{e}_{x'} - sin\,\theta\,\vec{e}_{z'}$

Après traitement nous obtenons :

$\vec{e}_{x''} = \cos\theta\cos\varphi\,\vec{e}_x + \cos\theta\sin\varphi\,\vec{e}_y - \sin\theta\,\vec{e}_z$ (27)

$\vec{e}_{y''} = -\sin\varphi\,\vec{e}_x + \cos\varphi\,\vec{e}_y$ (28)

$$\vec{e}_{z''} = \sin\theta\cos\varphi\,\vec{e}_x + \sin\theta\sin\varphi\,\vec{e}_y + \cos\theta\,\vec{e}_z \quad (29)$$

Puisque nous avons déjà l'équation cartésienne de la normale dans le repère fixe, nous pouvons déduire θ par produit scalaire entre $\vec{n}_u$ et $\vec{e}_z$, et l'angle φ par produit scalaire entre le projeté de $\vec{n}_u$ dans le plan $(\vec{e}_x, \vec{e}_y)$ et $\vec{e}_x$. On peut également utiliser à la place du projeté de $\vec{n}_u$ un vecteur qui lui est proportionnel comme le vecteur (X_n, Y_n).

6.2.2. Détermination de l'équation de la droite

Pour déterminer l'équation de la droite passant par l'axe de rotation de l'obstacle et de même orientation que le rayonnement incident dans le repère fixe de l'héliostat ombragé, nous allons d'abord déterminer cette équation dans le repère fixe de l'obstacle puis la transposer par translation dans le repère fixe de l'héliostat ombragé. L'équation d'une droite peut être déterminée par son vecteur directeur et un point d'application quelconque. Dans le repère fixe de l'obstacle, le point d'application est l'origine du repère soit le point de coordonnées (0 ; 0 ; 0). L'équation de la droite est alors simplement proportionnelle au vecteur du rayonnement incident déterminé p.23. Si un point M est élément de cette droite, alors $\vec{OM} = t \times \begin{pmatrix} X_{inc} \\ Y_{inc} \\ Z_{inc} \end{pmatrix}$

On obtient l'équation de cette droite dans le repère de l'héliostat ombragé par translation de vecteur $\vec{V}_{omb} = \begin{pmatrix} x_{omb} \\ y_{omb} \end{pmatrix}$ où x_{omb}, y_{omb} sont les coordonnées de l'obstacle dans le repère fixe de l'héliostat ombragé (les deux axes de rotation se trouvant à la même hauteur, l'obstacle n'a pas de troisième coordonnée). Dans ce repère, les coordonnées d'un point élément de la droite sont :

$$\vec{OM} = \begin{pmatrix} tX_{inc} + x_{omb} \\ tY_{inc} + y_{omb} \\ tZ_{inc} \end{pmatrix}$$

6.2.3. Détermination des coordonnées du plan

Dans le repère tournant associé à l'héliostat ombragé, l'équation du plan de ce dernier s'exprime simplement. En effet, si le point M est élément du plan, alors

$$\vec{O''M} = x''\vec{e}''_x + y''\vec{e}''_y$$

Soit puisque $O'' = O$ et en remplaçant $\vec{e}''_x$ *et* $\vec{e}''_y$ par leur coordonnées dans le repère fixe :

$$\vec{OM} = x''(\cos\theta\cos\varphi\,\vec{e}_x + \cos\theta\sin\varphi\,\vec{e}_y - \sin\theta\,\vec{e}_z) + y''(-\sin\varphi\,\vec{e}_x + \cos\varphi\,\vec{e}_y)$$

$$Soit\ \vec{OM} = (x''\cos\theta\cos\varphi - y''\sin\varphi)\,\vec{e}_x + (x''\cos\theta\sin\varphi + y''\cos\varphi)\,\vec{e}_y - x''\sin\theta\,\vec{e}_z \quad (30)$$

6.2.4. Détermination de l'intersection

Dans le repère fixe nous pouvons écrire : si un point M est à la fois élément de la droite et du plan on a l'égalité :

$$(x''\cos\theta\cos\varphi - y''\sin\varphi)\,\vec{e}_x + (x''\cos\theta\sin\varphi + y''\cos\varphi)\,\vec{e}_y - x''\sin\theta\,\vec{e}_z \quad (31)$$

Après traitement nous obtenons l'expression de $x''\ et\ y''$ soit directement les coordonnées de l'intersection dans le repère tournant attaché à l'héliostat.

$$x'' = \frac{y_{omb} + \dfrac{x_{omb}}{\tan\varphi}}{\cos\theta\sin\varphi + \sin\theta\dfrac{y_{inc}}{z_{inc}} + \dfrac{\cos\theta\cos\varphi + \sin\theta\dfrac{x_{inc}}{z_{inc}}}{\tan\varphi}} \quad (32)$$

$$y'' = [x''(\cos\theta\cos\varphi + \sin\theta\frac{x_{inc}}{z_{inc}}) - x_{omb}]\times\frac{1}{\sin\varphi} \quad (33)$$

Ainsi toutes les données nécessaires à la quantification de l'effet d'ombrage sont déterminées.

7. Quantification de l'effet de blocage

7.1. Détermination de la méthode

La mise en équation des effets de blocage présente une grande similarité avec la quantification des effets d'ombrage. Il s'agit essentiellement de la même démarche quelques différences près : par exemple la direction considérée pour déterminer la position des obstacles est l'angle au sol du rayon réfléchi. Cette direction est déterminée par la position de la tour relativement à l'héliostat. Si l'on nomme x_{tour} , y_{tour} les coordonnées de la tour, cet angle s'exprime par :

$$angle_{ref} = -\arctan\frac{y_{tour}}{x_{tour}} \quad (34)$$

Et nous retrouvons les conditions :

- $-90° < angle_{ref} \le -45°$: $x_{bloc1} = 0$, $y_{bloc1} = a$; $x_{bloc2} = a$, $y_{bloc2} = a$
- $-45° < angle_{ref} \le 0°$: $x_{bloc1} = a$, $y_{bloc1} = a$; $x_{bloc2} = a$, $y_{bloc2} = 0$
- $0° < angle_{ref} \le 45°$: $x_{bloc1} = a$, $y_{bloc1} = 0$; $x_{bloc2} = a$, $y_{bloc2} = -a$
- $45° < angle_{ref} \le 90°$: $x_{bloc1} = a$, $y_{bloc1} = -a$; $x_{bloc2} = 0$, $y_{bloc2} = -a$

Les équations de changement de repère restent évidemment les mêmes mais la détermination du projeté de l'axe de rotation de l'obstacle sur le plan de l'héliostat se fait cette fois suivant la direction du rayonnement réfléchi (et bien sûr dans le sens opposé).

7.2. Mise en équation

7.2.1. Détermination de l'équation de la droite

Nous déterminons l'équation de la droite de projection de manière similaire au cas des ombrages en remplaçant le vecteur incident par le vecteur $\vec{V}_{tour}$ des coordonnées de la tour relativement à l'héliostat.

Si M est élément de la droite les coordonnées du vecteur $\vec{OM}$ s'expriment par :

$$\vec{OM} = \begin{pmatrix} tX_{tour} + x_{bloc} \\ tY_{tour} + y_{bloc} \\ tZ_{tour} \end{pmatrix}$$

7.2.2. Détermination de l'équation du plan

L'équation du plan de l'héliostat est toujours la même, c'est-à-dire que si M est élément du plan, le vecteur $\vec{OM}$ s'exprime par :

$$\vec{OM} = (x''\cos\theta\cos\varphi - y''\sin\varphi)\,\vec{e}_x + (x''\cos\theta\sin\varphi + y''\cos\varphi)\,\vec{e}_y - x''\sin\theta\,\vec{e}_z$$

7.2.3. Détermination de l'intersection

Les coordonnées de l'intersection dans le repère tournant attaché à l'héliostat deviennent :

$$x'' = \frac{y_{bloc} + \dfrac{x_{bloc}}{\tan\varphi}}{\cos\theta\sin\varphi + \sin\theta\dfrac{y_{tour}}{z_{tour}} + \dfrac{\cos\theta\cos\varphi + \sin\theta\dfrac{x_{tour}}{z_{tour}}}{\tan\varphi}} \quad (35)$$

$$y'' = [x''(\cos\theta\cos\varphi + \sin\theta\frac{x_{tour}}{z_{tour}}) - x_{bloc}] \times \frac{1}{\sin\varphi} \quad (36)$$

Enfin l'effet de blocage, dans le cas simple de la figure 19 s'exprime par :

$$e_{omb} = 1 - \frac{(l - x_{inter}) \times (l - y_{inter})}{l^2} \quad (37)$$

Où x_{inter} *et* y_{inter} *représentent* x'' *et* y'' , ce qui quantifie les effets de blocage.

8. Prise en compte des superpositions

Pour évaluer correctement les effets d'ombrage et de blocage, il nous faut encore considérer le phénomène de superposition : il est en effet possible, suivant les dispositions, que deux obstacles génèrent des effets sur un même héliostat. Dans ce cas, on ne peut additionner directement les deux effets puisqu'il existe une zone de superposition. Nous devons donc distinguer les cas résumés à la figure 21 qui valent aussi bien pour le calcul des effets d'ombrage que celui des effets de blocage.

Pour distinguer toutes les combinaisons possibles, nous avons considéré les cas suivant :

- Les masques sont du même côté
  - Les deux masques sont à droites (cas a et b)
  - Les deux masques sont à gauche (cas c et d)
- Les masques sont de côtés opposés
  - Les deux masques se recoupent (cas e et f)
  - Les deux masques ne se recoupent pas (cas g et h)

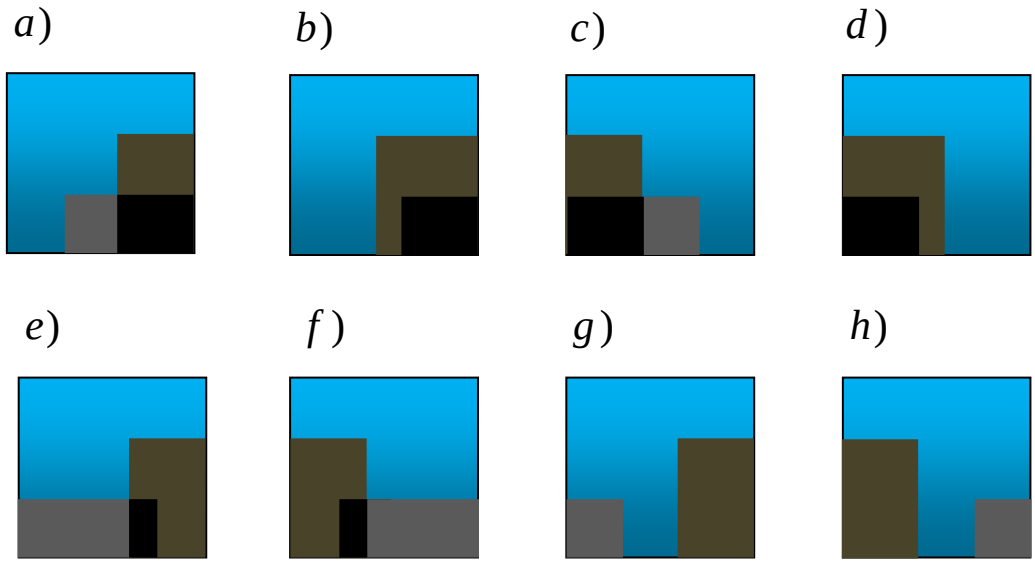


Figure 21 : illustration de toutes les configurations possibles pour la superposition de deux masques

Connaissant la position des deux obstacles, nous pouvons calculer les deux intersections $(x_{\text{int}er1}, y_{\text{int}er1})$ *et* $(x_{\text{int}er2}, y_{\text{int}er2})$. Il n'est pas difficile, en programmation, de déterminer quelle intersection génère l'ombrage le plus haut (en vert foncé sur la figure) et de nommer les coordonnées de l'intersection correspondante x_h, y_h . En nommant les coordonnées de la

deuxième intersection x_b, y_b, nous obtenons les formules de calcul présentées ci-dessous. Nous avons associé à ces formules les conditions qui permettront au programme de déterminer le cas de figure à considérer.

a) *Condition* : $y_h > 0,\ y_b > 0$ *et* $y_h > y_b$ $\quad Formule : e = 1 - \dfrac{(l - y_h)(l - x_h) + (y_h - y_b)(l - x_b)}{l^2}$

b) *Condition* : $y_h > 0,\ y_b > 0$ *et* $y_h < y_b$ $\quad Formule : e = 1 - \dfrac{(l - y_h)(l - x_h)}{l^2}$

c) *Condition* : $y_h < 0,\ y_b < 0$ *et* $y_h < y_b$ $\quad Formule : e = 1 - \dfrac{(l + y_h)(l - x_h) + (y_b - y_h)(l - x_b)}{l^2}$

d) *Condition* : $y_h < 0,\ y_b < 0$ *et* $y_h > y_b$ $\quad Formule : e = 1 - \dfrac{(l + y_h)(l - x_h)}{l^2}$

e) *Condition* : $y_h > 0,\ y_b < 0$ *et* $y_h - y_b < l$ $\quad Formule : e = 1 - \dfrac{(l - y_h)(l - x_h) + y_h \times (l - x_b)}{l^2}$

f) *Condition* : $y_h < 0,\ y_b > 0$ *et* $y_b - y_h < l$ $\quad Formule : e = 1 - \dfrac{(l + y_h)(l - x_h) - y_h \times (l - x_b)}{l^2}$

g) *Condition* : $y_h > 0,\ y_b < 0$ *et* $y_h - y_b > l$ $\quad Formule : e = 1 - \dfrac{(l - x_b)(l + y_b) + (l - y_h)(l - x_h)}{l^2}$

h) *Condition* : $y_h < 0,\ y_b > 0$ *et* $y_b - y_h > l$ $\quad Formule : e = 1 - \dfrac{(l - x_h)(l + y_h) + (l - y_b)(l - x_b)}{l^2}$

Muni de ces outils de calcul, le programme est à présent capable de calculer les matrices d'efficacité associées aux effets d'ombrage et de blocage. La situation se complique toutefois lorsque l'on doit regrouper ces deux effets pour afficher la matrice associée à l'effet ombrage + blocage qui doit permettre la comparaison avec Windelsol. Dans ce cas à nouveau il faut considérer le phénomène de superposition mais avec un nombre d'obstacles simultanés pouvant aller jusqu'à trois ou quatre. Le nombre de configurations possibles correspondant est trop important pour pouvoir traiter ce problème de la même manière que celle exposée ci-dessus. Pour cette raison, nous avons mis au point une technique permettant de calculer l'effet résultant sans nécessiter la détermination du cas de figure et quelques soit le nombre d'obstacles impliqués. Le principe est détaillé en annexe 5.

9. Présentation du logiciel

9.1. Aperçu de l'interface graphique

Il nous est impossible, dans le cadre d'un rapport de stage, d'exposer l'intégralité de la conception du logiciel. Cependant, le programme a bien été conçu suivant les lignes directrices exposées précédemment : le script est disponible en annexe 6. Le logiciel a été baptisé Uraeus. Dans la mythologie égyptienne, Uraeus représentait le cobra qui figurait sur la

coiffe des pharaons et symbolisait entre autres choses l'œil du dieu Rê (le dieu égyptien du soleil). C'est dorénavant par ce nom que nous y ferons référence. La figure 22 ci-dessous donne un aperçu de son interface graphique.

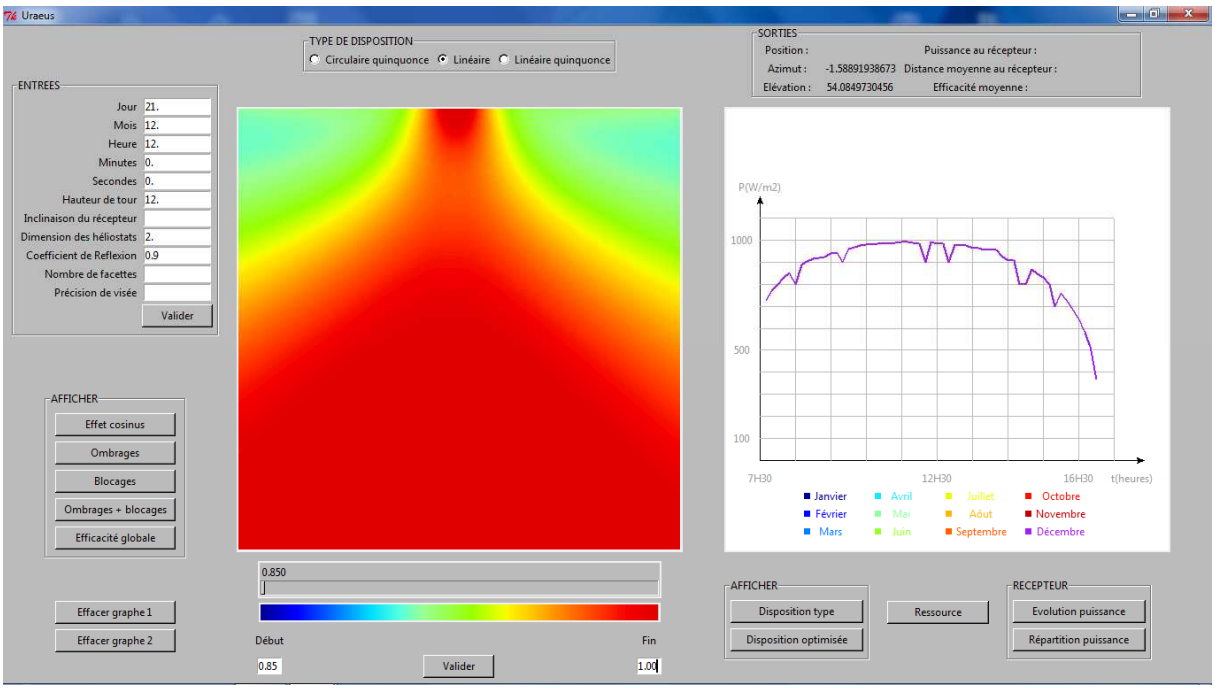


Figure 22 : Aperçu de l'interface graphique du logiciel Uraeus

L'interface est munie de deux zones de représentation graphique : le graphe de gauche permet d'afficher les matrices d'efficacité. Le graphe de droite peut afficher des données complémentaires de type différent. On peut choisir par exemple d'afficher la disposition des héliostats dans le champ solaire, l'évolution de la ressource solaire, l'évolution de la puissance au récepteur (donnée indispensable à l'optimisation du champ solaire, à la conception du bloc puissance où du récepteur lui-même). Au-dessus du graphe de gauche on propose à l'utilisateur le choix du type de disposition. Cependant parmi les trois types de dispositions proposés seuls deux sont fonctionnels : le type circulaire n'ayant été programmé qu'à titre comparatif, il n'est fonctionnel qu'aux alentours de midi solaire. En haut à gauche se trouve le cadre des entrées où doivent être renseignées toutes les données nécessaires aux calculs effectués par le logiciel c'est-à-dire la date précise, la hauteur de tour, l'inclinaison du récepteur, la dimension des héliostats, le coefficient de réflexion des héliostats, le nombre de facettes, la précision de visée. En dessous du cadre des entrées, des boutons de commande permettent d'afficher les différentes matrices d'efficacité sur le premier graphe. On trouve, en dessous de ce graphe une échelle de couleur modifiable en temps réel qui permet de visionner en détail l'amplitude de l'effet sur n'importe quelle partie du champ à n'importe quel instant.

En haut du graphe de droite se trouve le cadre associé aux sorties numériques qui renseigne notamment la position instantanée du soleil (élévation, azimut), la position d'un clic sur le premier graphe, la puissance instantanée au récepteur, la distance moyenne des héliostats au récepteur, l'efficacité moyenne des héliostats dans le champ solaire. Enfin on trouve sous le graphe de droite des boutons de commande permettant d'afficher la disposition type, la disposition optimisée, l'évolution journalière de la ressource solaire, l'évolution journalière du flux au récepteur, la répartition du flux au récepteur.

9.2. Premiers résultats : comparaison avec Windelsol

A titre comparatif, une disposition de type circulaire quinconce a été programmée. Cependant il ne s'agit pas d'une disposition absolument identique à celle utilisée par Windelsol (comme le montre la figure 22). De plus, nous ne disposons pas des paramètres qui ont été utilisés pour générer les matrices que nous souhaitons comparer (hauteur de tour, rayonnement incident...). Les effets ne seront donc comparés que du point de vue de la forme des matrices.

9.2.1. Les dispositions utilisées

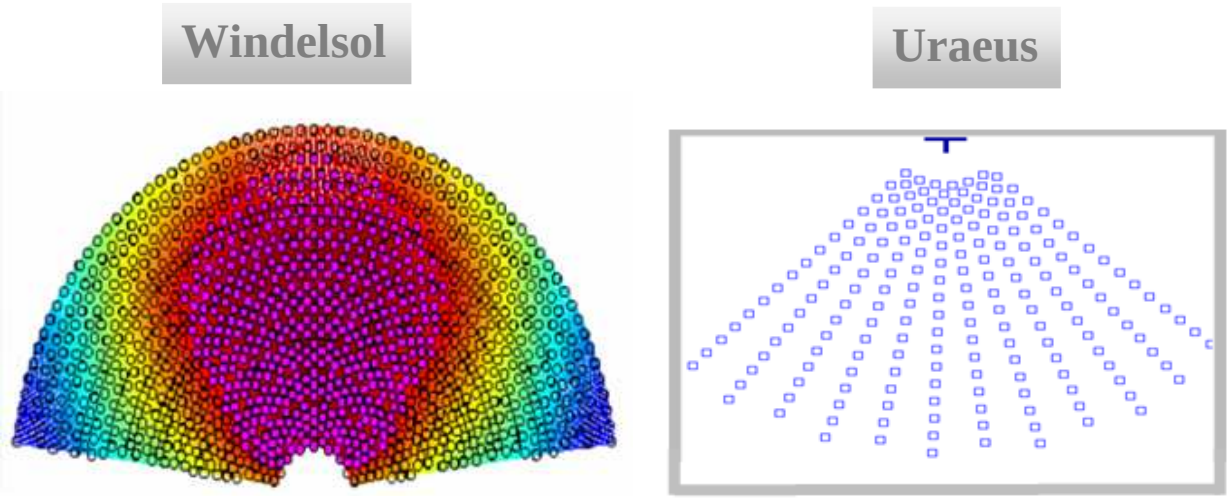


Figure 23 : Les deux dispositions utilisées par les logiciels sont du type circulaire quinconce

Les deux dispositions utilisées sont représentées sur la figure 22 (ci-dessus). Nous nous sommes efforcés de copier la disposition utilisée dans Windelsol (les points violets représentent les héliostats) qui est du type circulaire quinconce. On note cependant quelques différences : dans Windelsol, la distance entre deux rangées augmente légèrement au fur et à mesure que l'on s'éloigne de la tour (pour parer aux effets de blocage) alors qu'elle reste la même dans Uraeus. Dans Windelsol, sur une même rangée, l'angle séparant deux héliostats voisins diminue légèrement lorsque l'on s'éloigne de la tour alors que dans Uraeus il reste

constant. En résumé la disposition utilisée à titre comparatif dans Uraeus n'est pas une disposition optimisée, mais elle suffira à comparer les matrices d'efficacité du point de vue de la forme.

9.2.2. Comparaison de l'effet cosinus

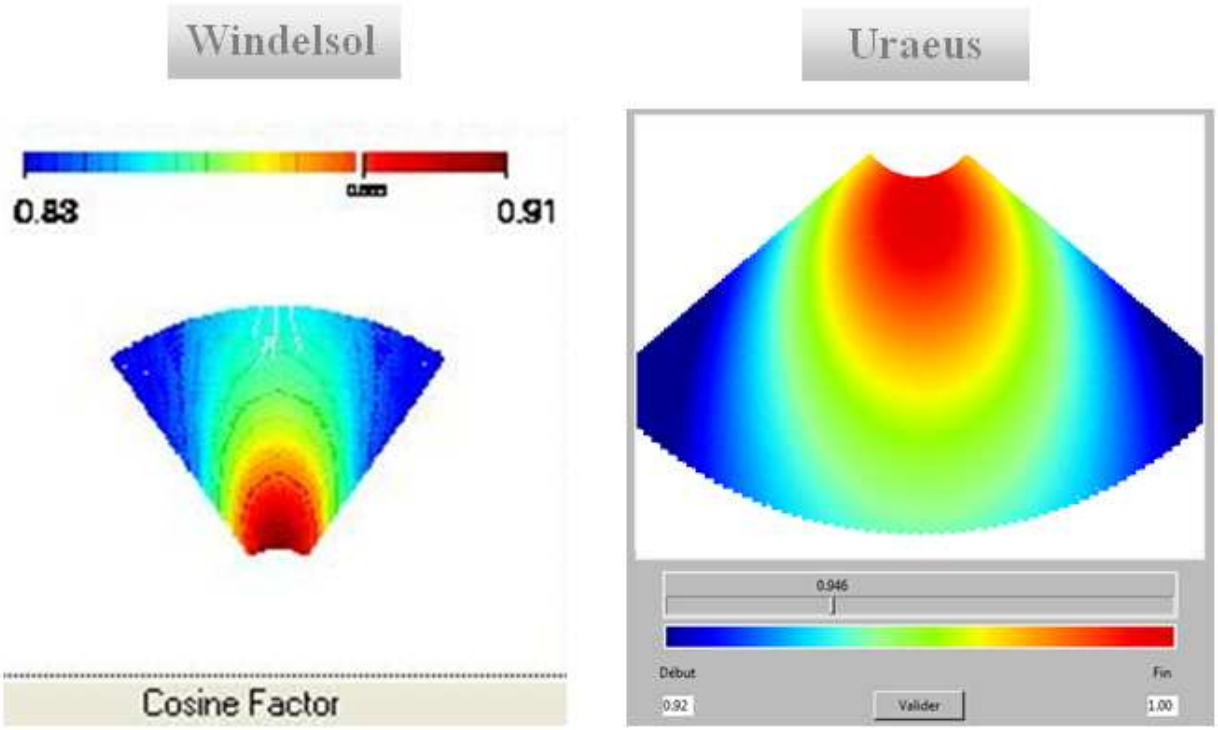


Figure 24 : Comparaison des matrices d'efficacité liées à l'effet cosinus calculées par les logiciels Windelsol et Uraeus

Comme nous l'avons vu précédemment, l'effet cosinus est le facteur de perte majoritaire dans un champ solaire pour centrale à tour. Sa valeur ne dépend pas de la disposition (contrairement aux effets d'ombrages et de blocages) mais seulement de paramètres externes comme l'inclinaison du rayonnement incident où la hauteur du récepteur. Il s'agit en conséquence de l'effet le plus simple à modéliser puisqu'aucune interaction avec d'autres héliostats du champ n'est à prendre en compte : en l'occurrence aucune approximation n'a été utilisée lors de la mise en équation. Sauf erreur dans la mise en équation (que la concordance des formes en comparaison à Windelsol semble exclure), l'effet cosinus a donc pu être modélisé de manière exacte.

9.2.3. Comparaison de l'effet associé ombrages et blocages

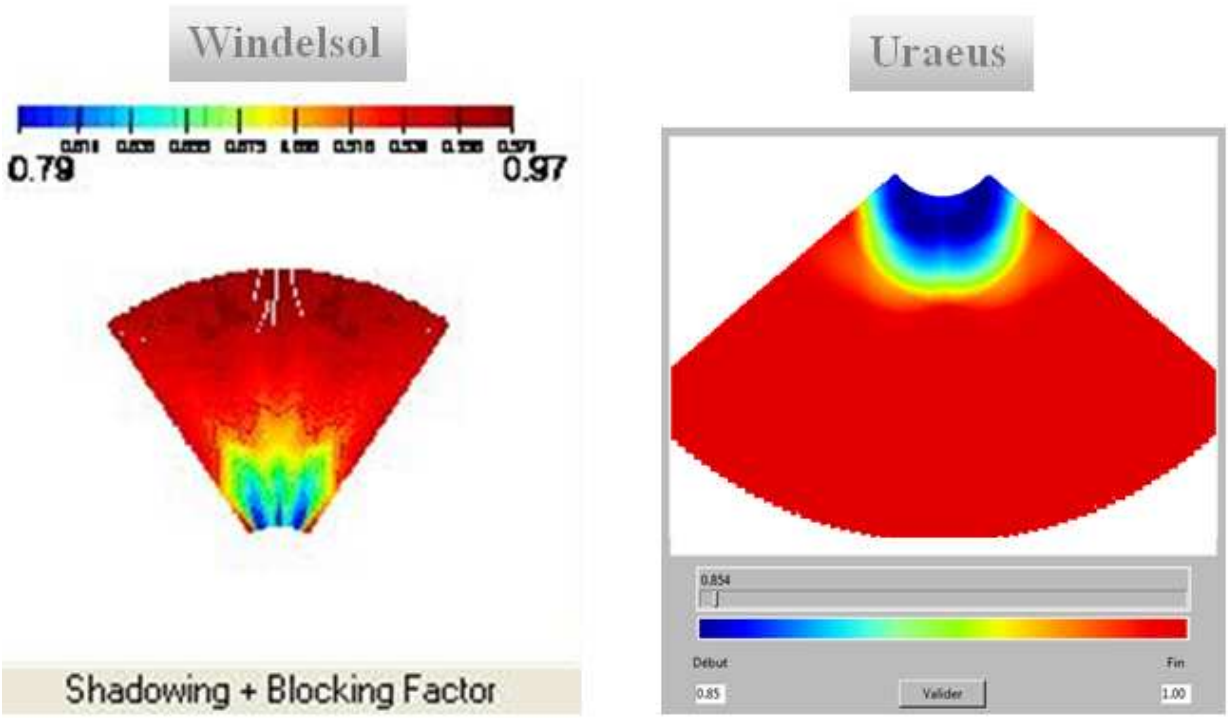


Figure 25 : Comparaison des matrices d'efficacité liées aux effets combinés d'ombrage et de blocage calculées par les logiciels Windelsol et Uraeus.

La modélisation des effets d'ombrages et de blocages fut l'étape la plus difficile dans la conception du logiciel. Il fallait en effet tenir compte de nombreuses interactions avec le voisinage qui se multipliaient encore avec la superposition des deux effets. La modélisation de ces effets fût rendue possible par la mise au point d'une technique d'élimination des superpositions (résumée en annexe 4).

La comparaison des formes semble assez concluante : l'effet est localisé près de la tour, on remarque deux protubérances latérales où l'effet est plus prononcé. Evidemment les matrices ne sont pas parfaitement similaires puisque les effets d'ombrage et de blocage sont dépendants de la disposition (contrairement à l'effet cosinus), et nous savons que la disposition utilisée par Uraeus n'est pas exactement la même que celle utilisée par Windelsol. De plus, nous ne connaissons pas les paramètres externes utilisés par Windesol (rayonnement incident, hauteur de récepteur, dimension du champ), nous ne pouvons donc pas affirmer avec aussi peu d'éléments la pertinence de la modélisation. Cependant, sur la base d'un mauvais traitement des équations ou d'une approximation inadaptée, il serait peu probable d'obtenir des matrices d'efficacité aussi proches en formes.

De part sa conception, Uraeus est capable d'afficher les effets d'ombrage et de blocage séparément. Une représentation des effets d'ombrage et de blocage dont la combinaison a été utilisé pour la comparaison (figure 24) est donnée figure 25.

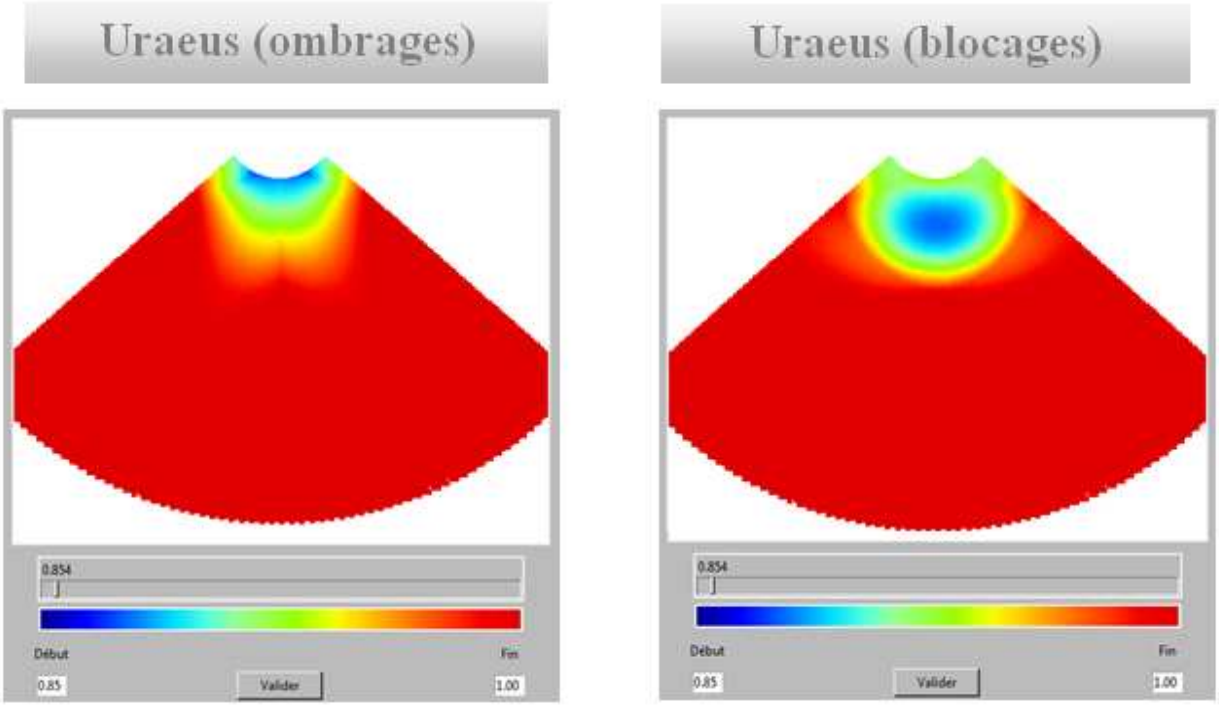


Figure 26 : matrices d'efficacité des effets d'ombrage et de blocage calculées par le logiciel Uraeus

10. Détermination de l'algorithme d'optimisation

10.1. Détermination des paramètres d'optimisation

La mise en place d'un algorithme d'optimisation adapté est une étape importante dans la conception du logiciel. Idéalement, l'algorithme devrait prendre en compte tous les paramètres, qualitatifs où quantitatifs, qui d'une manière où d'une autre influent sur la qualité de la disposition. Malheureusement un programme ne dispose pas de capacités de réflexion, et la mise au point d'un algorithme utilisant de nombreux paramètres est une opération périlleuse quant à la fiabilité des résultats. La plupart des logiciels disposent d'ailleurs d'algorithmes simples à l'instar d'HGM dont le programme d'optimisation ne prend en compte qu'un seul paramètre (l'efficacité).

Comme toujours il est essentiel de considérer les particularités du projet en question avant d'opérer un choix. L'objectif est ici d'utiliser un système d'entraînement commun aux héliostats et ainsi de réduire le nombre de moteurs et de systèmes de commande utilisés. Concrètement, l'opération nécessite un élément qui s'étale devant tous héliostats (le porche d'orientation), leur transmettant un mouvement commun qui est ensuite transformé par un

mécanisme simple au niveau de chaque héliostat. Sur le terrain des problèmes ont déjà été rencontrés quant à la flexibilité du porche. Bien que l'on puisse résoudre ces problèmes par l'utilisation d'une barre de meilleure qualité, il est évident que réduire la distance entre deux héliostats permettrait de diminuer la rigidité requise pour la barre du porche. De plus, la quantité de matériaux à utiliser pour assurer la mise en commun du système d'entraînement est proportionnelle à la distance moyenne séparant deux héliostats. Réduire la distance inter-héliostats semble donc une mesure pertinente tant d'un point de vue économique que pour la faisabilité de la construction mécanique.

Le projet de champ solaire synchronisé repose sur un concept nouveau : aucune réalisation à l'échelle d'une centrale à tour n'existe actuellement. Bien que le prototype conçu par Jeremy ZMUDA fonctionne sur le papier sans erreur d'angle, des imprécisions dans la construction mécanique induisent des erreurs de visée. Il en sera de même pour le champ solaire synchronisé puisqu'aucune construction mécanique ne peut prétendre à la perfection. A l'état d'avancement actuel du projet, il serait risqué de s'avancer sur la précision atteignable. Cette situation nous donne un argument supplémentaire pour privilégier la compacité du champ : si la distance moyenne des héliostats au récepteur est réduite, alors la précision de visée requise est réduite en proportion. D'un point de vue général un projet de central à tour est un projet complexe, et tout paramètre qui vise à en améliorer la faisabilité est à prendre en compte : dans notre cas la compacité du champ améliore la faisabilité et le prix de la construction mécanique et diminue la précision requise : la compacité sera donc un paramètre d'optimisation de la disposition.

Un deuxième paramètre que l'on peut considérer simultanément à la compacité est l'efficacité. En effet si les héliostats disposent d'une bonne efficacité, le nombre d'éléments requis pour une puissance donnée au récepteur n'est pas trop important et par suite le champ ne s'étale pas sur de grandes dimensions. A l'inverse, si l'efficacité devient très faible, le nombre d'héliostats requis augmente drastiquement de même que les dimensions du champ. Il existe donc une plage de valeurs où il doit être possible de corréler efficacité et compacité pour générer des dispositions présentant un ratio intéressant.

Il existe un troisième paramètre qui va influer sur la faisabilité du projet : l'évolution de la puissance au récepteur. Pour la conception du bloc de puissance (notamment pour l'ensemble turbine-compresseur), il est préférable que la puissance au récepteur soit la plus constante possible. La ressource solaire varie naturellement au cours de la journée et dans le cas d'une centrale à tour la variation de la puissance au récepteur s'amplifie avec les effets d'ombrages en début et en fin de journée comme le montre la figure 26.

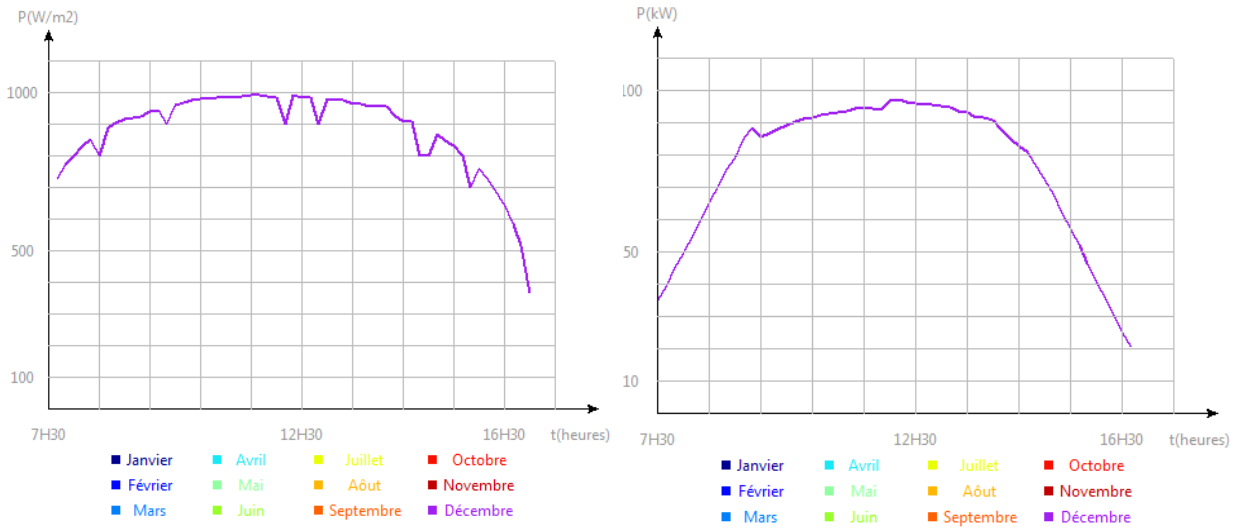


Figure 27 : Evolution de la ressource solaire, évolution de la puissance au récepteur pour une disposition type linéaire (non optimisée) représentées par le logiciel Uraeus

Une disposition capable de lisser cette variation faciliterait la conception du bloc de puissance. Néanmoins l'opération serait coûteuse en termes de compacité car pour minimiser les effets d'ombrages en début et fin de journée, la distance inter-héliostats devrait être augmentée de manière conséquente. L'évolution de la puissance au récepteur est donc certes un paramètre important mais il est trop difficile à prendre en compte simultanément à la compacité et l'efficacité. D'autres solutions pourront être trouvées lors la conception du bloc de puissance. Par exemple l'utilisation complémentaire de carburant pourrait compléter les irrégularités de la puissance au récepteur. On pourrait également utiliser plusieurs turbines de faibles puissances que l'on chargerait en fonction de l'apport au récepteur, ou un système permettant de rentrer ou sortir plus où moins l'aimant de l'alternateur de son bobinage, ce qui aurait pour effet de faire varier le couple sur l'arbre moteur.

Finalement les paramètres que nous décidons de considérer sont la compacité et l'efficacité.

10.2. Fonctionnement de l'algorithme

Nous ne décrirons dans cette partie que le fonctionnement de l'algorithme, le script associé est donné à la fin du script du logiciel (voir annexe 5).

Il est important de remarquer qu'à l'instar des logiciels existants, la disposition est optimisée à midi solaire : une disposition optimisée dans la matinée présenterait par exemple une efficacité trop diminuée l'après-midi et inversement. Une conséquence directe (exploitée dans la suite) d'une optimisation à midi solaire est la symétrie de la disposition générée.

L'algorithme d'optimisation effectue 5 étapes successives :

1) Détermination de la position de la première ligne
2) Génération de la disposition sur la première ligne

Pour les lignes suivantes :

3) Détermination de la position de la ligne
4) Disposition des héliostats sur la ligne
5) Symétrisation

La première étape s'effectue par un calcul itératif : le logiciel considère une disposition linéaire type et y calcul l'efficacité moyenne des héliostats. Il décale ensuite entièrement le champ solaire et l'éloigne légèrement de la tour (+10 cm). A nouveau l'efficacité moyenne est calculée et la même procédure est répétée jusqu'à atteindre un éloignement maximum prédéterminé dans le script (la première ligne à 5 mètres de la tour). La figure 27 illustre le procédé. Le logiciel compare ensuite les efficacités obtenues et choisis comme position pour la première ligne la distance à la tour permettant les meilleures performances. La position de la première ligne est alors déterminée.

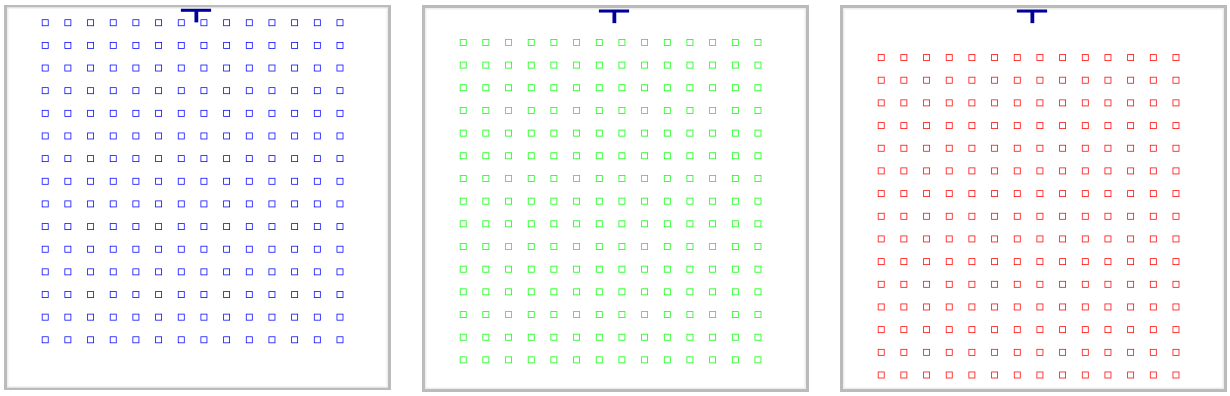


Figure 28 : Uraeus décale la disposition et effectue des calculs comparatifs déterminant ainsi le meilleur emplacement pour la première ligne

La deuxième étape consiste à disposer les héliostats sur la première ligne. Le logiciel les place par comparaison avec une efficacité minimum spécifique à la deuxième étape. Le logiciel considère une position puis calcule le cumul de l'effet cosinus et des effets de blocages (il n'y a pas d'ombrages pour la première ligne). Si l'efficacité résultante est supérieure ou égale à l'efficacité minimale, l'héliostat est placé et sa position mémorisée dans une liste. Dans le cas contraire une nouvelle position légèrement décalée (+20 cm) est utilisée pour un nouveau calcul jusqu'à obtenir l'efficacité minimum et ainsi de suite jusqu'à ce que toute la ligne ait été parcourue.

Une fois la première ligne entièrement déterminée le logiciel passe à la troisième étape : il génère une ligne légèrement plus éloignée de la tour où les héliostats sont positionnés en quinconce par rapport aux héliostats de la ligne précédente, puis calcule l'efficacité moyenne. Si cette efficacité est supérieure où égale à l'efficacité minimum fixée pour la troisième étape, le logiciel mémorise la position de la ligne dans une liste et l'opération s'arrête. Dans le cas contraire, les calculs sont réitérés pour une ligne un peu plus éloignée (+20 cm) et ainsi de suite jusqu'à atteindre l'efficacité minimum et déterminer la position de la ligne.

La quatrième étape permet au logiciel de placer les héliostats sur la ligne précédemment déterminée suivant une efficacité minimum. Les héliostats sont d'abord placés de gauche à droite et les calculs sont effectués en prenant en compte les obstacles de devant et l'obstacle gauche (l'obstacle de droite n'existant pas encore). Il résulte de cette situation que seule la partie droite (Ouest) de la disposition est générée de manière pertinente, ce qui nous amène à la cinquième étape.

La dernière étape consiste simplement à symétriser la disposition par rapport à sa partie droite (côté Ouest) et donc à générer une liste où cette partie est recopiée à gauche (côté Est).

Une fois la première ligne déterminée, la répétition en boucle des étapes 3, 4 et 5 permet de générer la disposition des héliostats dans le champ. A la fin de chaque ligne, Uraeus calcule la puissance totale au récepteur. Lorsque celle-ci dépasse 300 kW (ou toute autre valeur fixée par l'utilisateur), les calculs s'arrêtent et la disposition est enregistrée.

Cette approche utilisée pour générer des dispositions présente plusieurs avantages : premièrement il est laissé à l'utilisateur le choix de trois paramètres (4 avec la puissance au récepteur) pour influer sur la disposition. En diminuant l'efficacité minimum associée à la deuxième étape, on augmente par exemple la compacité et l'étendue de la première ligne. L'efficacité associée à la troisième étape permet d'influer sur l'espacement entre les lignes et celle associé à la quatrième étape permet de modifier la compacité des héliostats sur les lignes suivantes. Par action sur ces 3 paramètres l'utilisateur pourra donc faire varier le ratio compacité/efficacité jusqu'à obtenir le compromis qu'il jugera le meilleur. On note que le nombre d'héliostats par ligne n'a pas à être renseigné, il est automatiquement optimisé par l'efficacité minimum associée.

V. RESULTATS

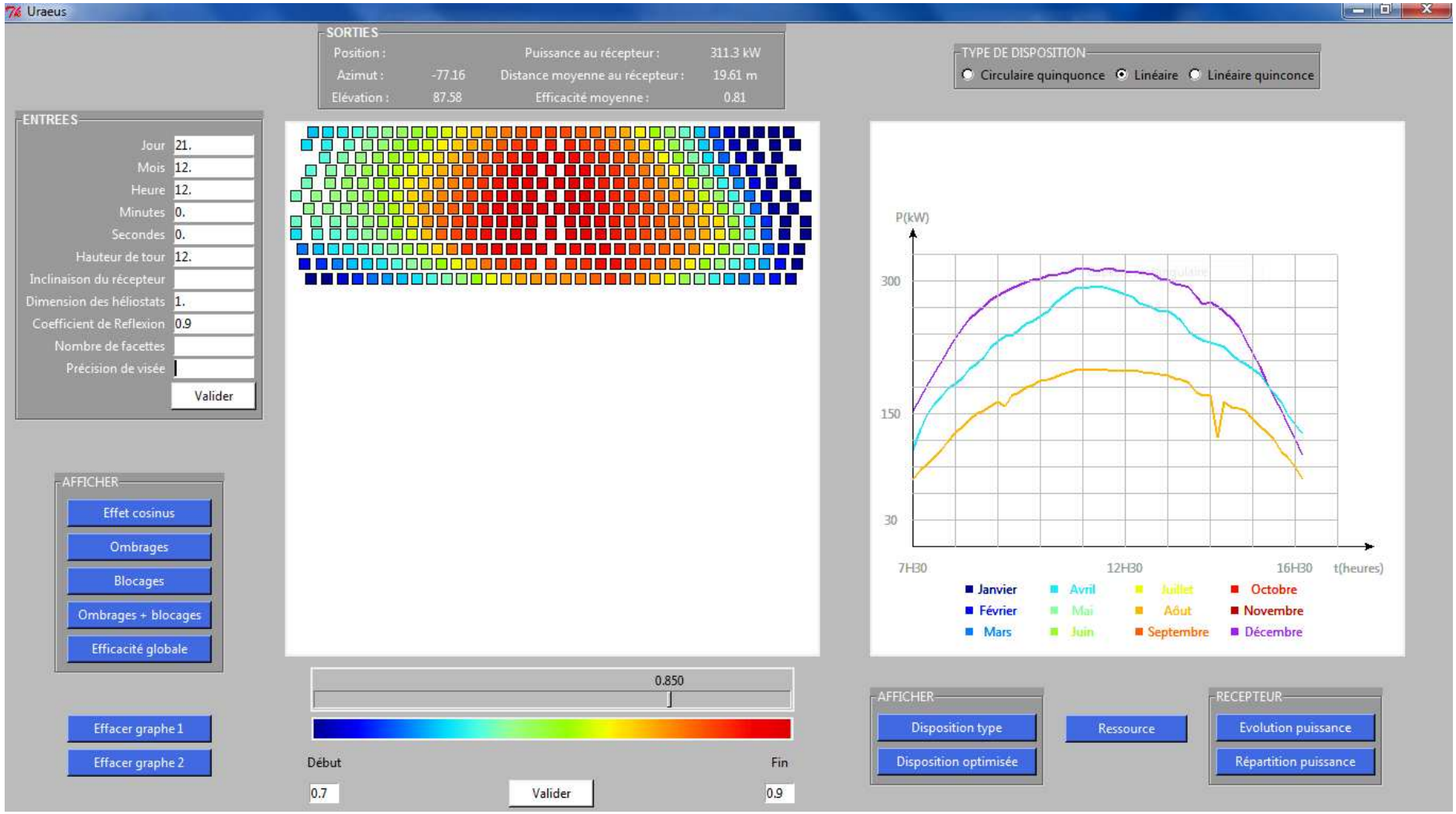


Figure 29 : disposition optimisée par Uraeus, pour des héliostats de 1 mètre de côté.

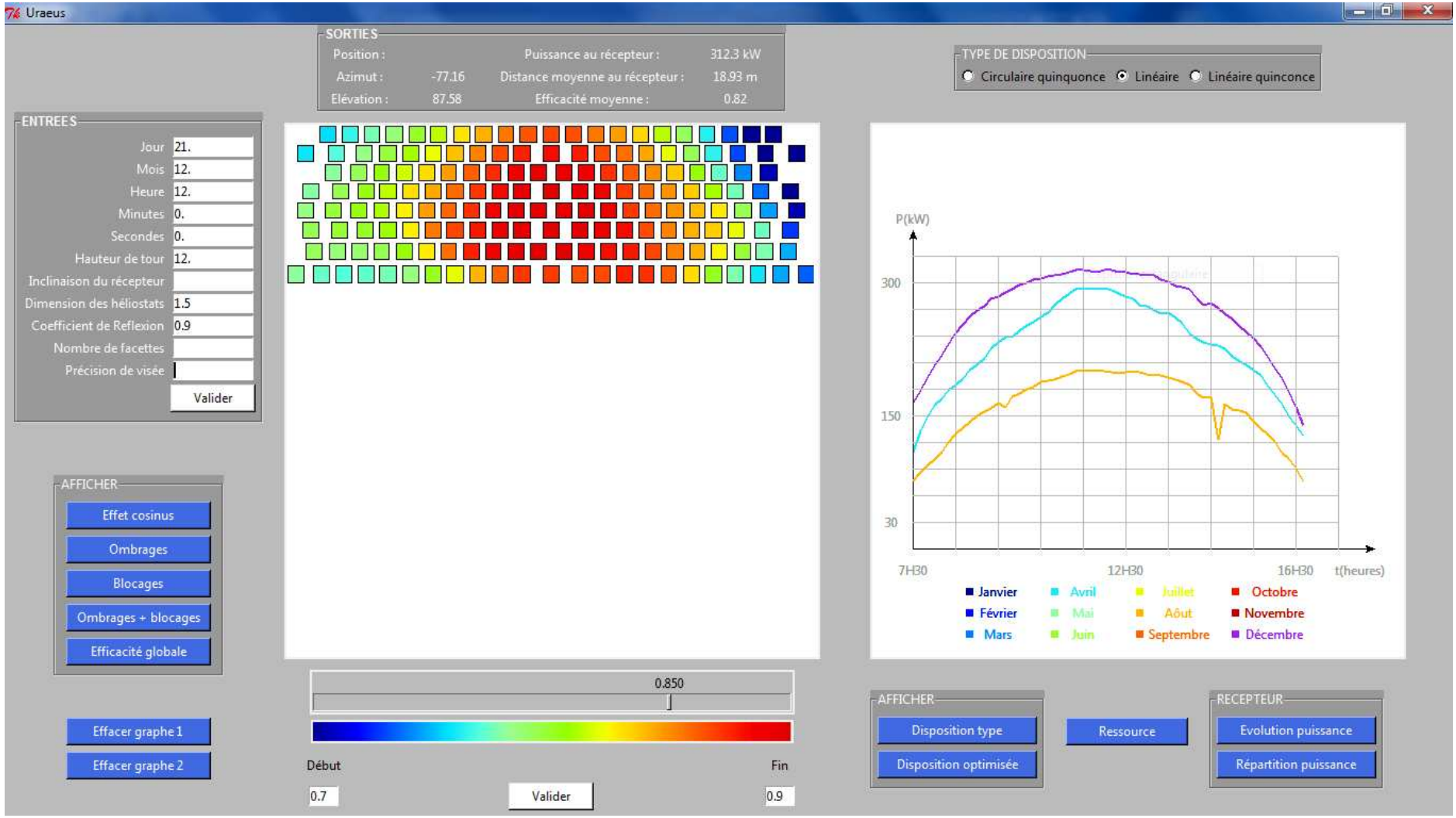


Figure 30 : disposition optimisée par Uraeus, pour des héliostats de 1,5 mètre de côté.

VI. DISCUSSION ET ANALYSES

Les résultats exposés sur les figures 29 et 30 représentent les dispositions qui du point de vue de l'utilisateur conduisaient au meilleur ratio compacité/efficacité. Dans ces dispositions chaque héliostat est rempli d'une couleur exprimant son efficacité totale. Sur le graphe de gauche est représentée l'évolution de la puissance au récepteur, tracée pour les mois de décembre, avril et août. Uraeus intègre en effet une partie des données météorologiques relevées par les stations de Kamboinse au cours de l'année 2010. Chaque mois une journée parfaitement ensoleillée a été choisie et la valeur du DNI relevée toutes les 10 minutes.

Pour les deux dispositions la distance moyenne des héliostats au récepteur est très faible (inférieure à 20 mètres). La précision requise pour le champ solaire est donc largement diminuée, ce qui favorise la faisabilité du projet. L'efficacité moyenne s'élève à 81-82%, ce qui peut être considéré comme excellent mais il ne faut pas oublier que cette efficacité est évaluée à midi solaire au cours du mois le plus favorable c'est-à-dire décembre. D'un point de vue général, l'efficacité décroit fortement en début et fin de journée et plus faiblement au cours des mois (en rapport à décembre). Le fait que les meilleurs résultats soient obtenus en décembre est du à la faible valeur de l'élévation à midi solaire (environ 55°). Cet angle est proche de celui sous lequel les héliostats voient le récepteur, en conséquence l'effet cosinus (qui est l'effet majoritaire dans le champ solaire) est largement diminué. De plus le DNI est maximum en décembre, ce qui est probablement du à la faible valeur de l'humidité au cours de ce mois. Les deux dispositions présentées ne diffèrent (du point de vue des paramètres d'optimisation) que par la taille des héliostats. D'après les résultats obtenus, la disposition associée aux héliostats de 2,25 m² (1,5 mètre de côté) présente une meilleure compacité car la distance moyenne des héliostats au récepteur est légèrement inférieure. Il faut ajouter à cela que le nombre d'héliostats nécessaire est également réduit (de 388 à 171), ce qui peut être un avantage pour les délais de réalisation du champ solaire. La prise au vent est en revanche un facteur défavorable aux héliostats de grandes dimensions (les héliostats de petites dimensions s'abritant mutuellement). Pendant la saison pluvieuse ou en février, le site de Kamboinse est régulièrement soumis à des vents violents. Ce paramètre est donc à prendre en compte dans le choix de la dimension des héliostats. Deux autres dispositions où la hauteur de tour a été légèrement diminuée sont données en annexe 6. Elles illustrent bien l'influence de la hauteur du récepteur (soit indirectement l'influence de l'effet cosinus et des blocages) sur la compacité de la disposition.

VII. CONCLUSION ET PERSPECTIVES

Au terme de ce travail, nous pouvons conclure que sur les trois échelles d'optimisation associées au champ solaire, deux ont pu être déterminées (la disposition des héliostats dans le champ solaire et la dimension des héliostats). L'optimisation de la troisième échelle (la taille des facettes) nécessite la programmation de la fonctionnalité Soltrace, qui n'a pas pu être réalisée dans les délais de l'étude. La programmation de cette fonctionnalité, en plus de permettre cette dernière optimisation, donnerait au logiciel Uraeus un sérieux argument comparé aux logiciels existants (notamment Windelsol) où l'utilisation complémentaire de Soltrace est nécessaire pour déterminer la répartition de la puissance au niveau du récepteur. Bien que Soltrace génère des résultats très précis, ce logiciel n'a pas été développé pour l'étude des champs solaires associés aux centrales à tours. En particulier chaque simulation effectuée impose la reconfiguration intégrale (et manuelle) de la disposition et/ou de l'orientation de tous les héliostats du champ solaire. Cette contrainte limite fortement le nombre de simulations effectuées et de dispositions testées, ce qui nuit à la qualité de l'optimisation. Enfin il est possible d'utiliser cette fonctionnalité pour évaluer de manière exacte tous les facteurs de pertes associés au champ solaire. Si la comparaison des matrices d'efficacité ainsi générées avec celles présentées dans ce mémoire était concluante, nous validerions non seulement les mises en équations et l'approximation utilisée, mais également une approche originale et beaucoup plus simple pour modéliser les facteurs de pertes des champs solaires pour centrale à tour.

Au regard des résultats obtenus, bien que les dispositions générées présentent des performances intéressantes, elles pourraient probablement encore être améliorées (en particulier la distance moyenne des héliostats au récepteur) par la génération de dispositions s'étendant des deux côtés de la tour (Nord et Sud). En effet le projet de centrale à tour du 2ie est le premier projet de ce type au monde à être situé sous les tropiques (très faibles latitudes). Dans ces conditions il est possible d'étendre le champ du côté Sud de la tour sans être trop pénalisé par l'effet cosinus et ainsi diminuer la distance moyenne des héliostats au récepteur. On peut remarquer également que dans de telles configurations la puissance au récepteur varierait de façon moins brutale en début où fin de journée puisqu'une partie du champ (Nord ou Sud suivant les périodes de l'année) serait exempt d'ombrages.

Les résultats sont perfectibles, mais ce travail a permis de montrer que le 2ie peut développer ses propres outils pour mener à termes son projet de centrale à tour. De belles perspectives pour le logiciel, et indirectement pour les dispositions générées, sont envisageables.

REFERENCES

[1] www.sonabel.bf/perspec/electrif_bf.htm

[2] Numerical simulation of a flow field in a friction type turbine (Tesla turbine), Andrés Felipe Rey LADINO, Institute of Thermal Powerplants Vienna University of Technology, 14 juin 2004.

[3] Solar Field Efficiency and Electricity Generation Estimations for a Hybrid Solar Gas Turbine Project in France, Pierre GARCIA, Alain FERRIERE, Gilles FLAMAND, 7 janvier 2008.

[4] Methodology for generation of heliostat field layout in central receiver systems based on yearly normalized energy surfaces, Marcelino SANCHEZ, Manuel ROMERO, 27 juillet 2005.

[5] Modélisation des centrales solaires thermodynamiques, Cas du projet PEGASE à THEMIS, Pierre GARCIA, Alain FERRIERE, Stéphane PLAYS, Jean-Jacques BEZIAN, 1 juin 2007.

[6] Les centrales solaires à tour, enseignement dispensé par Alain FERRIERE niveau master2 à Odeillo, 2010.

VIII. ANNEXES

SOMMAIRE DES ANNEXES

ANNEXE 1 : GENERALITES THERMODYNAMIQUES

Le fonctionnement d'une centrale thermique est basé sur la propriété d'extension d'un fluide lorsqu'il est chauffé. Lors d'un apport thermique, la distance entre les molécules qui composent le fluide augmente, ce qui a pour effet de générer une force d'expansion. C'est cette force d'expansion qu'on exploite dans un moteur ou une turbine. Cependant une partie de l'énergie absorbée par les molécules est convertie en énergie de vibration : la température augmente. Pendant la détente du gaz et suite aux collisions entre molécules, une partie de cette énergie vibratoire est convertie en énergie cinétique (puis en travail par le piston), mais il est statistiquement improbable que l'ensemble des molécules reconvertissent l'intégralité de leur énergie vibratoire en énergie cinétique. Ainsi on ne peut convertir l'intégralité de la chaleur en travail, c'est la raison pour laquelle un gaz est toujours plus chaud en fin de détente qu'en début de compression. Dans la plupart des centrales thermiques, un condenseur est donc nécessaire pour évacuer cette chaleur non convertie.

Une centrale thermique fonctionne sur le principe illustré par la figure 1.

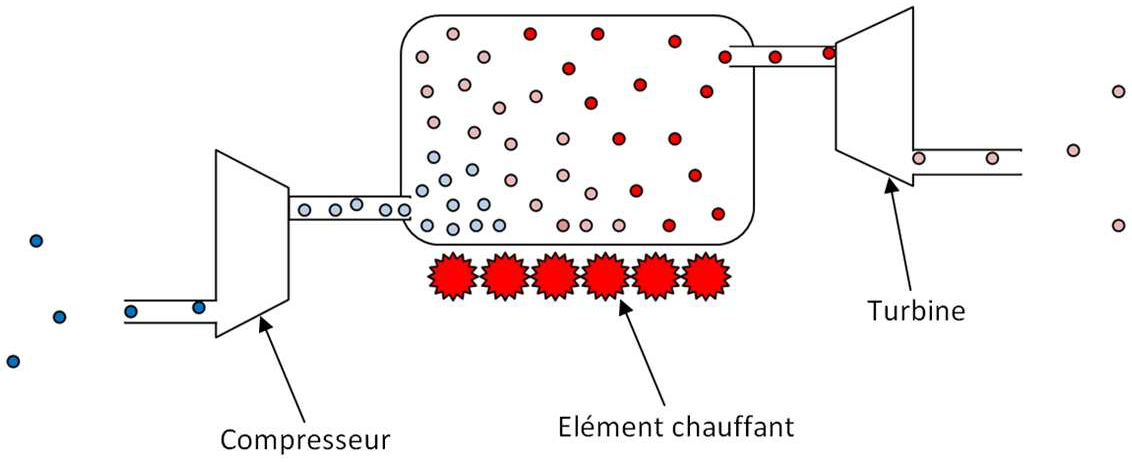


Figure 31 : schéma d'une centrale thermique fonctionnant suivant un cycle de Brayton

La turbine est solidaire d'un alternateur. C'est l'écoulement du fluide au travers de celle-ci qui entraîne l'ensemble et pour qu'un fluide s'écoule dans une turbine, la pression d'entrée doit être supérieure à la pression sortie. L'analogie souvent faite entre la chaudière

d'une centrale thermique et une cocotte-minute est trompeuse. En effet, elle laisse imaginer que le rôle de la chaleur est de faire monter le fluide en pression. Si la pression d'un fluide que l'on chauffe dans un espace confiné augmente effectivement, la production d'une centrale thermique basée sur ce principe serait intermittente car il faudrait attendre que la chaudière se vide pour pouvoir ensuite la recharger. Une telle contrainte de production est bien sûr inenvisageable, c'est la raison pour laquelle un élément qui recharge en permanence la chaudière en fluide pressurisé est nécessaire. Le rôle de l'apport de chaleur dans la chaudière n'est donc pas de faire monter le fluide en pression mais de générer une poussée supplémentaire à celle due à l'écart de pression (généré par le compresseur). On récupère ainsi en sortie de turbine plus d'énergie qu'on en a utilisé pour comprimer le fluide. Le fonctionnement d'une centrale électro-solaire ne diffère de celui d'une centrale thermique classique que part la façon d'apporter la chaleur dans la chaudière. Si dans la deuxième on utilise des carburants fossiles, on obtient un résultat similaire en utilisant dans la première un rayonnement solaire concentré. Un des principaux intérêts du cycle de Brayton est qu'il permet de se passer du refroidissement : en fin de détente, on peut rejeter l'air chaud dans l'environnement et aspirer l'air frais qui nous entoure. L'utilisation d'un cycle de Brayton permet donc d'éliminer un élément onéreux des centrales thermiques : le condenseur. Puisque le refroidissement n'est plus nécessaire, toute consommation d'eau est également éliminée. Le cycle de Brayton présente toutefois des contraintes, notamment la température élevée à laquelle l'apport de chaleur doit être réalisé. En effet si l'on peut pratiquement négliger l'augmentation de température d'un liquide que l'on comprime (cycle de Rankine), ce n'est pas le cas lors de la compression d'un gaz. Si l'on souhaite ensuite chauffer ce gaz comprimé, la source de chaleur utilisée doit automatiquement permettre d'atteindre des températures bien supérieures à celle du gaz en fin de compression, c'est-à-dire 500°C au minimum. Pour réaliser un cycle de Brayton par voie solaire, des concentrations élevées doivent donc être utilisées, les centrales à tour permettent d'atteindre de telles concentrations.

ANNEXE 2 : FONCTIONNEMENT DE HGM

HGM est un code nouvelle génération développé par CIEMAT. Comme son nom l'indique, ce code fait grandir le champ solaire héliostat par héliostat, en plaçant à chaque fois le nouvel héliostat à l'emplacement le plus favorable. Il est en effet possible, pour une hauteur de tour donnée, de calculer l'énergie annuellement disponible au récepteur selon l'emplacement qu'occupe l'héliostat sur le site. HGM découpe le site en grille et après avoir testé toutes les dispositions possibles en tenant compte des ombrages ou blocages générés par les autres éléments, il place l'héliostat à la position la plus favorable. A l'instar de Rcell qui avait été développé par l'Université de Houston en 1978, HGM découpe le site en grille dont les rangées sont circulaires et centrées sur le récepteur.

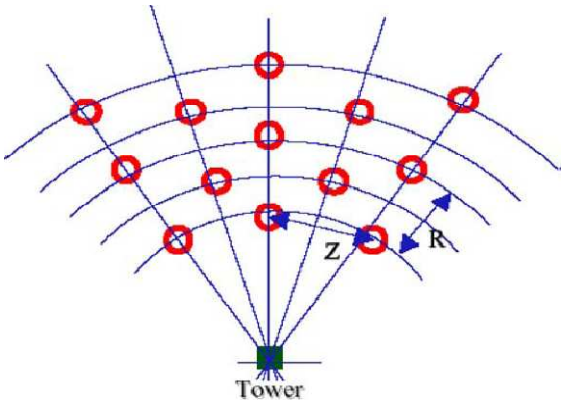


Figure 32 : Représentation de la découpe du champ en grille par HGM

En plus de la disposition optimale, HGM est capable d'afficher des cartes représentant l'amplitude des pertes pour les différents facteurs (blocages, ombrages, effet cosinus).
Une représentation est donnée figure 8.

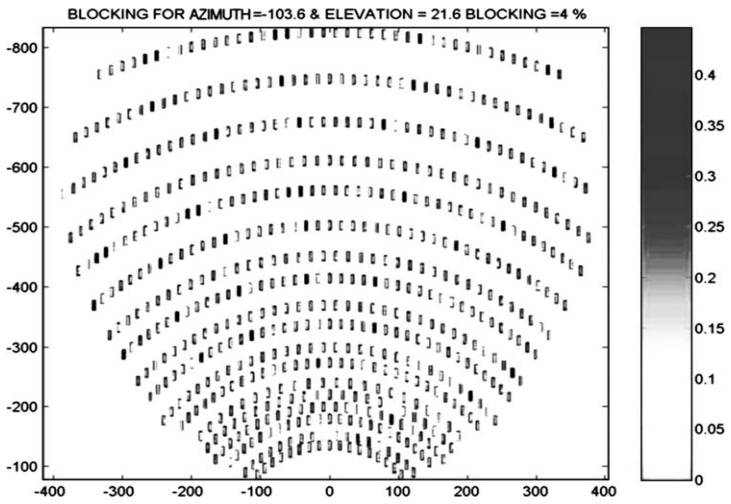


Figure 33 : Représentation de l'effet de blocage pour un champ d'héliostats généré par HGM

ANNEXE 3 : LOGICIELS EXISTANT

Tableau 1 : comparatif des logiciels existant

Etude comparative sur les codes pour le calcul des flux concentrés

Name	MIRVAL	FIAT LUX	WINDELSOL	HFLCAL	SOLTRACE
Utilisé par	SANDIA / DLR	CIEMAT-PSA	CIEMAT	DLR	NREL
Technologies	CRS	CRS	CRS	CRS	CRS-PT-DS-SF-PV
Années de développement	1978	1999-2000 (pas de manuel utilisateur)	2001	1986	1999-2004
Disponibilité	public	en validation	public	non diffusé	public
Langage de Prog.	FORTRAN	MATLAB	FORTRAN - Basic	FORTRAN	Delphi5
Méthode de calcul	Monte-Carlo	Monte-Carlo (?)	Convolution	Convolution	Monte-Carlo
Réflexion multiple	oui	non	non	oui	oui
Type de surface réceptrice	plane ou cylindrique	plane	plane, cavité ou cylindrique	plane, cylindrique ou conique	quelconque
Modèle pour performances annuelles	oui	non, calcul à une instant donné	oui	oui	non, calcul à une instant donné
Calculs économiques	non	non	oui	oui	non
Optimisation	sur critères énergétiques (ajout DLR)	non	par méthode RS, sur critères énergétiques	sur critères économiques	non

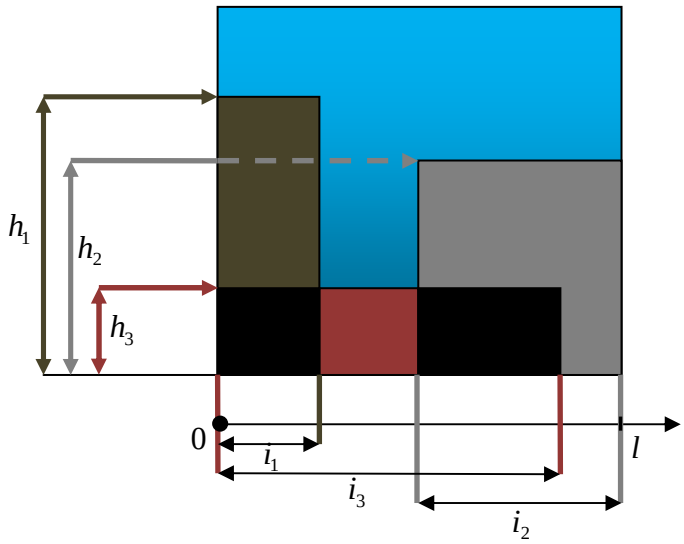
ANNEXE 4 : TECHNIQUE D'ELIMINATION DES SUPERPOSITIONS

La technique utilisée pour calculer l'effet de superposition des ombrages et des blocages répond à une des principales difficultés liée à la programmation du logiciel, il nous a semblé nécessaire de fournir des explications sur le principe utilisé.

Nous nous baserons sur un exemple simplifié et ne rentrerons pas dans tous les détails des traitements préalables nécessaires à l'élaboration des données utilisées.

Nous utilisons pour notre calcul deux données : la hauteur des masques et le début ainsi que la fin des intervalles correspondants. Ces données sont obtenues à partir des coordonnées de l'intersection associée à chaque masque et quelques calculs simples combinés à l'utilisation de listes et de dictionnaires.

Comprendre cette technique de programmation n'est pas primordial, l'essentiel est de retenir le résultat : les hauteurs sont classées dans une liste par ordre décroissant et les intervalles correspondant sont classés dans une seconde liste de même longueur. Ainsi nous disposons des données résumées par le schéma ci-dessous.



Nous supposerons dans notre exemple que la valeur de l (le côté de l'héliostat) est égale à l'unité, nous utilisons les valeurs :

$h_1 = 0.79,\ h_2 = 0.63,\ h_3 = 0.26$

$i_1 = [0;0.2],\ i_2 = [0.5;1],\ i_3 = [0;0.8]$

Dans notre exemple, nous commençons par multiplier les bornes des intervalles par dix pour transformer les décimaux en entiers, les intervalles deviennent:

$i_1 = [0;2],\ i_2 = [5;10],\ i_3 = [0;8]$

Nous transformons ensuite ces intervalles en listes comportant chacune 10 éléments. Ces

éléments ont pour valeur 1 si le numéro de l'élément dans la liste est compris dans l'intervalle correspondant, 0 sinon. Nous obtenons :

$i_1 = [1,1,0,0,0,0,0,0,0,0]$, $i_2 = [0,0,0,0,0,1,1,1,1,1]$, $i_3 = [1,1,1,1,1,1,1,1,0,0]$

Le but de la technique est de comptabiliser au fur et à mesure les aires occupées par les masques en éliminant au préalable les zones de recouvrement par comparaison avec une liste mémorisant les zones occupées par tous les masques de hauteur supérieur, systématiquement mise à jour (il est en effet facile, avec un logiciel de programmation comme python, de parcourir plusieurs listes éléments par éléments tout en effectuant des comparaisons entre ceux-ci).

Pour le premier masque, cette liste (référencée sous le nom de $zone_{occupée}$) n'existe pas encore, on peut donc écrire :

$aire_1 = \frac{somme[i_1]}{10} \times h_1 = \frac{2}{10} \times 0,79 = 0,158$ *où* $somme[i_1]$ représente la somme de tous les éléments de la liste i_1 (équivalent à la longueur de l'intervalle du masque associé multipliée par 10).

La liste de mémorisation est alors mise à jour : $zone_{occupée} = [1,1,0,0,0,0,0,0,0,0]$

Nous effectuons à présent deux opérations simultanées : le programme compare élément par élément la liste suivante (i_2) et la liste de mémorisation ($zone_{occupée}$), si $zone_{occupée}[i] - i_2[i] = -1$ il met à jour l'élément concerné dans la liste $zone_{occupée}$ *c'est à dire* : $zone_{occupée}[i] = 1$. Dans le cas contraire, il remplace l'élément $i_2[i]$ par 0 pour créer la liste i_2' correspondant au masque exempt des zones d'intersection avec les masques de hauteurs supérieures. Ainsi on obtient :

$zone_{occupée} = [1,1,0,0,0,1,1,1,1,1]$

$i_2' = [0,0,0,0,0,1,1,1,1,1]$ (puisqu'il n'y a pas d'intersection entre i_2 *et* i_1).

Nous avons donc $aire_2 = \frac{somme[i_2']}{10} \times h_2 = \frac{5}{10} \times 0,63 = 0,315$

La même démarche est effectuée pour tous les intervalles successifs :

La deuxième mise à jour de la liste de mémorisation donne : $zone_{occupée} = [1,1,1,1,1,1,1,1,1,1]$

Et la liste i_3 devient : $i_3' = [0,0,1,1,1,0,0,0,0,0]$

Ainsi $aire_3 = \frac{somme[i_3']}{10} \times h_3 = \frac{3}{10} \times 0,26 = 0,078$

L'aire totale est la somme de toutes les aires soit :

$aire_{tot} = aire_1 + aire_2 + aire_3 = 0{,}158 + 0{,}315 + 0{,}078 = 0{,}551$ ce qui correspond bien à l'aire masquée. On remarque qu'à aucun moment dans la démarche nous n'avons eu à spécifier de cas de figure : cette technique est valable pour toutes les configurations possibles. De plus, grâce aux fonctionnalités dont dispose python sur les types de données évoluées comme les listes et les dictionnaires, il a été possible de programmer une fonction effectuant ces calculs sans même connaître le nombre d'obstacles.

ANNEXE 5 : SCRIPT DU LOGICIEL

```
#################################################################
# Programme Python                                              #
# Auteur: Emmanuel Clerc                                        #
# Nom du logiciel: Uraeus                                       #
# Fonction: Optimiser la configuration d'un champ solaire       #
# Réalisé en qualité de: Stagiaire au LESEE                     #
# Lieu: Ouagadougou                                             #
# Date: 2011                                                    #
#################################################################


# Importation des modules externes nécessaires au programme:
from math import*
from Tkinter import*

# Pour obtenir un dégradé de qualité, la palette de couleur qui suit a été
# sélectionnée manuellement. Il s'agit des seuls couleurs utilisées par le code.


coul=['#000090','#000098','#0000a0','#0000a8','#0000b0','#0000b8','#0000c0',
      '#0000c8','#0000d0','#0000d8','#0000e0','#0000e8','#0000f0','#0000f8',
      '#0000ff','#0008ff','#0010ff','#0018ff','#0020ff','#0028ff','#0030ff',
      '#0038ff','#0040ff','#0048ff','#0050ff','#0058ff','#0060ff','#0068ff',
      '#0070ff','#0078ff','#0080ff','#0088ff','#0090ff','#0098ff','#00a0ff',
      '#00a8ff','#00b0ff','#00b8ff','#00c0ff','#00c8ff','#00d0ff','#00d8ff',
      '#00e0ff','#08e3fc','#10e6f9','#18e9f6','#20ecf3','#28eff0','#30f3ed',
      '#38f6ea','#40f9e7','#48fce3','#50ffe0','#58ffdd','#60ffd0','#68ffc8',
      '#70ffc0','#78ffb8','#80ffb0','#88ffa8','#90ffa0','#98ff98','#99ff90',
      '#99ff88','#99ff80','#99ff78','#99ff70','#99ff68','#99ff60','#99ff58',
      '#99ff50','#99ff48','#99ff40','#99ff38','#99ff30','#99ff28','#99ff20',
```

```
        '#b8ff00','#c0ff00','#c8ff00','#d0ff00','#d8ff00','#e0ff00','#e8ff00',
        '#ebff00','#eeff00','#fcf800','#fef200','#ffec00','#ffe600','#ffe100',
        '#ffdd00','#ffd900','#ffd400','#ffcd00','#ffc800','#ffc200','#ffbc00',
        '#ffb600','#ffb000','#ffaa00','#ffa500','#ff9e00','#ff9800','#ff9300',
        '#ff8d00','#ff8800','#ff8300','#ff7c00','#ff7800','#ff7000','#ff6b00',
        '#ff6500','#ff6000','#ff5c00','#ff5600','#ff5000','#ff4c00','#ff4600',
        '#ff4000','#ff3c00','#ff3700','#ff3100','#ff2c00','#ff2700','#ff2000',
        '#fb1b00','#f71800','#f71000','#f70b00','#f70800','#f50000','#f30000',
        '#f10000','#f00000','#ee0000','#ec0000','#ea0000','#e80000','#e60000',
        '#e40000','#e20000','#e00000']


################################################################################
#                            INTERFACE GRAPHIQUE                               #
# Cette première partie du code est destinée à créer l'interface graphique au  #
# travers de laquelle l'utilisateur va interagir avec le code.                 #
################################################################################

# Création d'une fenêtre:
fen=Tk()
fen.title("Uraeus")
fen.configure(bg='#bbbbbb')

# Création d'un canevas utilisé pour les représentations graphiques, d'un
# deuxième canevas et d'une échelle utilisés pour la légende :
canevas=Canvas(fen,width=500,height=500,bg='white')
canevas.grid(row=6,column=2,rowspan=30,columnspan=5,padx=10,pady=10)
canevas2=Canvas(fen,width=500,height=500,bg='white')
canevas2.grid(row=6,column=8,columnspan=4,rowspan=30,padx=10,pady=10)
canevas3=Canvas(fen,width=450,height=18,bg='#bbbbbb')
canevas3.grid(row=37,column=2,columnspan=5)
echelle=Scale(fen,length=450,orient=HORIZONTAL,sliderlength=5.,
from_=0.85,to=1.,resolution=0.001,bg='#bbbbbb',troughcolor='#bbbbbb')
echelle.grid(row=36,column=2,columnspan=5)
```

```
echelle=Scale(fen,length=450,orient=HORIZONTAL,sliderlength=5.,
from_=0.85,to=1.,resolution=0.001,bg='#bbbbbb',troughcolor='#bbbbbb')
echelle.grid(row=36,column=2,columnspan=5)
# Grâce à une boucle while, on parcours l'espace du canevas2 en changeant
#la couleur de remplissage au fur et à mesure
i=0
while i<=149:
    canevas3.create_rectangle(3*i,0,3*(i+1),20,outline=coul[i],fill=coul[i])
    i+=1


# Création des variables Tkinter correspondant aux zones de saisie:
jour,mois,heure,minute,seconde,hauteur,inclinaison,taille,reflexion,precision,\
facette,disposition,debut,fin=StringVar(),StringVar(),StringVar(),\
StringVar(),StringVar(),StringVar(),StringVar(),StringVar(),StringVar(),\
StringVar(),StringVar(),StringVar(),StringVar(),StringVar()
disposition.set(0)

#Création des étiquettes configurables ou contenant d'autres Labels:
entrees=LabelFrame(fen,text='ENTREES',bg='grey60',fg='white',font=('President',9,'bold'),padx=5,pady=5)
sorties=LabelFrame(fen,text='SORTIES',bg='grey60',fg='white',font=('President',9,'bold'))
selectiondispo=LabelFrame(fen,text='TYPE DE DISPOSITION',bg='grey60',fg='white')
afficher=LabelFrame(fen,text='AFFICHER',bg='grey60',fg='white',padx=10,pady=5)
afficher2=LabelFrame(fen,text='AFFICHER',bg='grey60',fg='white',padx=5,pady=5)
recepteur=LabelFrame(fen,text='RECEPTEUR',bg='grey60',fg='white',padx=5,pady=5)
altitud=Label(sorties,width=12,bg='grey60',fg='white')
azim=Label(sorties,width=12,bg='grey60',fg='white')
etiquette1=Label(sorties,bg='grey60',fg='white',width=12)
etiquette2=Label(sorties,bg='grey60',fg='white',width=12)
etiquette3=Label(sorties,bg='grey60',fg='white',width=12)
pointeur=Label(sorties,width=12,fg='white',bg='grey60')
```

```
# Création des étiquettes fixes:
Label(entrees,text='Jour ',bg='grey60',fg='white').grid(row=2,column=0,sticky=E)
Label(entrees,text='Mois ',bg='grey60',fg='white').grid(row=3,column=0,sticky=E)
Label(entrees,text='Heure ',bg='grey60',fg='white').grid(row=4,column=0,sticky=E)
Label(entrees,text='Minutes ',bg='grey60',fg='white').grid(row=5,column=0,sticky=E)
Label(entrees,text='Secondes ',bg='grey60',fg='white').grid(row=6,column=0,sticky=E)
Label(entrees,text='Hauteur de tour ',bg='grey60',fg='white').grid(row=7,column=0,sticky=E)
Label(entrees,text='Inclinaison du récepteur ',bg='grey60',fg='white').grid(row=8,column=0,sticky=E)
Label(entrees,text='Dimension des héliostats ',bg='grey60',fg='white').grid(row=9,column=0,sticky=E)
Label(entrees,text='Coefficient de Reflexion ',bg='grey60',fg='white').grid(row=10,column=0,sticky=E)
Label(entrees,text='Nombre de facettes ',bg='grey60',fg='white').grid(row=11,column=0,sticky=E)
Label(entrees,text='Précision de visée ',bg='grey60',fg='white').grid(row=12,column=0,sticky=E)
Label(fen,bg='#bbbbbb',width=3).grid(row=14,column=7)
Label(fen,text='Début',bg='#bbbbbb',width=8).grid(row=38,column=2)
Label(fen,text='Fin',bg='#bbbbbb',width=8).grid(row=38,column=6)
Label(fen,bg='#bbbbbb',width=10).grid(row=39,column=3)
Label(fen,bg='#bbbbbb',width=10).grid(row=39,column=5)
Label(sorties,text='Position :',bg='grey60',fg='white',width=10).grid(row=2,column=8,sticky=E)
Label(sorties,text='Azimut :',bg='grey60',fg='white',width=10).grid(row=3,column=8,sticky=E)
Label(sorties,text='Elévation :',bg='grey60',fg='white',width=10).grid(row=4,column=8,sticky=E)
Label(sorties,text='Puissance au récepteur :',bg='grey60',fg='white',width=25).grid\
                            (row=2,column=10,sticky=E)
Label(sorties,text='Distance moyenne au récepteur :',bg='grey60',fg='white',width=25).grid\
                           (row=3,column=10,sticky=E)
Label(sorties,text='Efficacité moyenne :',bg='grey60',fg='white',width=25).grid\
                            (row=4,column=10,sticky=E)


# Création des zones de saisie:
Entry(entrees,textvariable=jour,width=12).grid(row=2,column=1)
Entry(entrees,textvariable=mois,width=12).grid(row=3,column=1)
Entry(entrees,textvariable=heure,width=12).grid(row=4,column=1)
Entry(entrees,textvariable=minute,width=12).grid(row=5,column=1)
Entry(entrees,textvariable=seconde,width=12).grid(row=6,column=1)
```

```
Entry(entrees,textvariable=hauteur,width=12).grid(row=7,column=1)
Entry(entrees,textvariable=inclinaison,width=12).grid(row=8,column=1)
Entry(entrees,textvariable=taille,width=12).grid(row=9,column=1)
Entry(entrees,textvariable=reflexion,width=12).grid(row=10,column=1)
Entry(entrees,textvariable=precision,width=12).grid(row=11,column=1)
Entry(entrees,textvariable=facette,width=12).grid(row=12,column=1)
Entry(fen,textvariable=debut,width=4).grid(row=39,column=2)
Entry(fen,textvariable=fin,width=4).grid(row=39,column=6)

# Positionnement des étiquettes configurables ou contenant d'autres Labels:
entrees.grid(row=2,column=0,rowspan=19,columnspan=2,padx=10)
sorties.grid(row=1,column=2,columnspan=5)
selectiondispo.grid(row=0,column=8,rowspan=6,columnspan=4)
afficher.grid(row=20,column=0,rowspan=18,columnspan=2)
afficher2.grid(row=36,column=8,rowspan=10)
recepteur.grid(row=36,column=10,rowspan=10)
pointeur.grid(row=2,column=9,sticky=W)
altitud.grid(row=4,column=9,sticky=W)
azim.grid(row=3,column=9,sticky=W)
etiquette1.grid(row=2,column=11,sticky=W)
etiquette2.grid(row=3,column=11,sticky=W)
etiquette3.grid(row=4,column=11,sticky=W)

# Cette première fonction est liée à l'interface graphique et en particulier
# aux boutons radio définis ci-dessous: son rôle est de changer l'action des
# boutons en fonction de la disposition choisie
def recupv():
    global choix
    if eval(disposition.get())==1:
        choix='circulaire'
    elif eval(disposition.get())==2:
        choix='linéaire'
    elif eval(disposition.get())==3:
        choix='linéaire quinquonce'
```

```
# Instantiation des boutons et boutons radio:
b1=Button(entrees,text='Valider',width=10,bg='white',fg='black')
b2=Button(afficher,text=" Effet cosinus ",width=18,bg='royal blue',fg='white',\
          command=lambda ar1=0, ar2=0,ar3=0,ar4=0:trace(0,0,0,0))
b3=Button(afficher,text=" Ombrages ",width=18,bg='royal blue',fg='white',\
          command=lambda ar1=1,ar2=0,ar3=0,ar4=0:trace(1,0,0,0))
b4=Button(afficher,text=" Blocages ",width=18,bg='royal blue',fg='white',\
          command=lambda ar1=0,ar2=1,ar3=0,ar4=0:trace(0,1,0,0))
b5=Button(afficher,text=" Ombrages + blocages ",width=18,bg='royal blue',fg='white',\
          command=lambda ar1=0,ar2=0,ar3=1,ar4=0:trace(0,0,1,0))
b6=Button(afficher,text=" Efficacité globale ",width=18,bg='royal blue',fg='white',\
          command=lambda ar1=0,ar2=0,ar3=0,ar4=1:trace(0,0,0,1))
b7=Button(fen,text="Effacer graphe 1",width=18,bg='royal blue',fg='white')
b72=Button(fen,text="Effacer graphe 2",width=18,bg='royal blue',fg='white')
b8=Button(afficher2,text=" Disposition type ",width=20,bg='royal blue',fg='white')
b9=Button(afficher2,text='Disposition optimisée',width=20,bg='royal blue',fg='white')
b10=Button(recepteur,text='Evolution puissance',width=20,bg='royal blue',fg='white')
b11=Button(recepteur,text='Répartition puissance',width=20,bg='royal blue',fg='white')
b12=Button(fen,text="Valider",width=10,bg='white',fg='black')
b13=Button(fen,text='Ressource',width=15,bg='royal blue',fg='white')
r1=Radiobutton(selectiondispo,text='Circulaire quinquonce',bg='grey60',fg='black',\
variable=disposition,value=1,command=recupv)
r2=Radiobutton(selectiondispo,text='Linéaire',bg='grey60',fg='black',variable=disposition\
,value=2,command=recupv)
r3=Radiobutton(selectiondispo,text='Linéaire quinconce',bg='grey60',fg='black',\
variable=disposition,value=3,command=recupv)

# Positionnement des boutons et des boutons radio dans la fenêtre:
b1.grid(row=13,column=1,pady=3)
b2.grid(row=15,pady=3)
b3.grid(row=17,pady=3)
b4.grid(row=19,pady=3)
b5.grid(row=21,pady=3)
b6.grid(row=23,pady=3)
b7.grid(row=37,column=0,columnspan=2,pady=3)
b72.grid(row=38,column=0,columnspan=2,pady=3)
```

```
b72.grid(row=38,column=0,columnspan=2,pady=3)
b8.grid(row=37,column=8,pady=3)
b9.grid(row=38,column=8,pady=3)
b10.grid(row=19,column=5,pady=3)
b11.grid(row=20,column=5,pady=3)
b12.grid(row=39,column=4)
b13.grid(row=37,column=9)
r1.grid(row=1,column=2)
r2.grid(row=1,column=3)
r3.grid(row=1,column=4)

###############################################################################
#                        DEFINITION DES FONCTIONS                             #
# Le code a été décomposé en fonctions:chacune d'elles est capable d'effectuer #
# certaines tâches et peut être appelée par d'autres fonctions, elles même     #
# executées lors d'un clic sur un bouton de l'interface graphique. Suivant les #
# choix réalisés, seules certaines fonctions seront executées.                 #
###############################################################################

def coordonnees(event):
    # En un clic de souris sur la carte, cette fonction permet d'afficher
    # les coordonnées du lieu
    pointeur.configure(text='x='+str((event.x-250)/10.)+' y='+str((event.y)/10.))


def position(event):
    # Cette fonction calcul la position du soleil une fois tous les paramètres
    # d'entrée renseignés. Elle est executée lors d'un clic sur le bouton
    # 'valider les données' de l'interface graphique.
    global azimut,altitude,h,l,deb,fi,ref,moi,heur,minut,variable
    variable=0
    num=275*eval(mois.get())/9-2*(eval(mois.get())+9)/12+eval(jour.get())-30
    horaireloc=eval(heure.get())+eval(minute.get())/60+eval(seconde.get())/3600
    h,l,deb,fi,ref,moi=eval(hauteur.get()),eval(taille.get()),eval(debut.get()),\
    eval(fin.get()),eval(reflexion.get()),eval(mois.get())
    heur,minut=eval(heure.get()),eval(minute.get())
```

```
    TSM=horaireloc-1.558/15
    positionabs=360*(num-1)/365.242
    eot=0.258*cos(positionabs*pi/180)-7.416*sin(positionabs*pi/180)-3.648*\
    cos(2*positionabs*pi/180)-9.228*sin(2*positionabs*pi/180)
    TSV=TSM+eot/60
    declinaison=(asin(0.39795*cos(0.98563*(num-173)*pi/180)))
    w=(TSV-12)*15
    altitude=(asin(sin(declinaison)*sin(12.459*pi/180)+cos(declinaison)*\
    cos(w*pi/180)*cos(12.459*pi/180)))*180/pi
    azimutn=(acos((sin(declinaison)*cos(12.459*pi/180)-cos(declinaison)*\
    cos(w*pi/180)*sin(12.459*pi/180))/cos(altitude*pi/180)))*180/pi
    if sin(w*pi/180)>0:
        azimut=180-azimutn
    else:
        azimut=-180+azimutn
    altitud.configure(text=str(int(altitude*100)/100.))
    azim.configure(text=str(int(azimut*100)/100.))
    echelle.configure(from_=deb,to=fi)


# La première disposition programmée est la disposition linéaire, nous nous
# efforcerons de toujours conserver le schéma suivant: une fonction calcul
# l'effet, une fonction choisis les couleurs (en rapport aux valeurs de l'effet)
# et une fonction parcours la zone graphique et trace l'effet.

def effetCos():
    # Quelque soit le type de disposition choisie, le calcul de l'effet cosinus
    # est toujours effectué par la même fonction.
    global eff,x,y,xsol,ysol,zsol,xtour,ytour,htour,xn,yn,zn
    if variable==1:
        angle1,angle2=azimu,alt
    else:
        angle1,angle2=azimut,altitude
    if angle1<-90 or angle1>90:
        xsol=-1/sqrt((1+tan(angle1*pi/180)**2)*(1+tan(angle2*pi/180)**2))
```

```
    else:
        angle1,angle2=azimut,altitude
    if angle1<-90 or angle1>90:
        xsol=-1/sqrt((1+tan(angle1*pi/180)**2)*(1+tan(angle2*pi/180)**2))
    else:
        xsol=1/sqrt((1+tan(angle1*pi/180)**2)*(1+tan(angle2*pi/180)**2))
    ysol=-xsol*tan(angle1*pi/180)
    zsol=tan(angle2*pi/180)*sqrt(xsol**2+ysol**2)
    xtour,ytour,htour=y,x-250.,h*10
    c2=htour/sqrt(xtour**2+ytour**2)
    c1=ytour/xtour
    xref=1/sqrt(1+c1**2+c2**2+c1**2*c2**2)
    yref=xref*c1
    zref=c2*sqrt(xref**2+yref**2)
    xn,yn,zn=xref+xsol,yref+ysol,zref+zsol
    eff=(xn*xsol+yn*ysol+zn*zsol)/sqrt(xn**2+yn**2+zn**2)

def couleur():
    # Fonction qui permet au programme de choisir la couleur en fonction
    # de la valeur de l'effet cosinus
    global c
    if variable2==0:
        if etot<=deb:
            c=coul[0]
        elif etot==fi:
            c=coul[149]
        else:
            x1=deb
            i=1
            while i<149:
                if etot>x1 and etot<=x1+(fi-deb)/149:
                    c=coul[i]
                i+=1
                x1+=(fi-deb)/149
```

```
    else:
        if eff<=deb:
            c=coul[0]
        elif eff==fi:
            c=coul[149]
        else:
            x1=deb
            i=1
            while i<149:
                if eff>x1 and eff<=x1+(fi-deb)/149:
                    c=coul[i]
                i+=1
                x1+=(fi-deb)/149


# Nous évaluons à présent l'effet d'ombrage pour une disposition linéaire. Comme
# il est possible que 2 héliostats ombrage en même temps, nous devons d'abord
# déterminer le nombre d'obstacles

def nombreOmbr():
# Fonction qui calcul le nombre et la position des obstacles ombrageant ainsi
# que d'autres données nécessaires au calcul de l'effet
    global signal,xh,yh,xb,yb,xinter,yinter
    global dico,ombloc
    if variable==1:
        angle=azimu
    else:
        angle=azimut
    dico={}
    ombloc=[]
    teta=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    a=1.2
    if choix=='linéaire'and variable2==1:
        if angle>=-90 and angle<-45:
            xomb1,yomb1,xomb2,yomb2=0,2*l,2*l,2*l
```

```
    elif angle>=-45 and angle<0:
        xomb1,yomb1,xomb2,yomb2=2*l,2*l,2*l,0
    elif angle>=0 and angle<45:
        xomb1,yomb1,xomb2,yomb2=2*l,0,2*l,-2*l
    else:
        xomb1,yomb1,xomb2,yomb2=2*l,-2*l,0,-2*l
if choix=='linéaire'and variable2==0:
    if angle>=-90 and angle<-45:
        xomb1,yomb1,xomb2,yomb2=0,y1,xobst,y2
    elif angle>=-45 and angle<0:
        xomb1,yomb1,xomb2,yomb2=xobst,y2,xobst,y3
    elif angle>=0 and angle<45:
        xomb1,yomb1,xomb2,yomb2=xobst,y2,xobst,y3
    else:
        xomb1,yomb1,xomb2,yomb2=xobst,y3,xobst,y4
if choix=='linéaire'and variable2==2:
    if angle>=-90 and angle<-45:
        xomb1,yomb1,xomb2,yomb2=10*l,-l,0,yb2
    elif angle>=-45 and angle<0:
        xomb1,yomb1,xomb2,yomb2=10*l,-l,10*l,-l
    elif angle>=0 and angle<45:
        xomb1,yomb1,xomb2,yomb2=10*l,-l,10*l,-l
    else:
        xomb1,yomb1,xomb2,yomb2=10*l,-l,0,-yb2
elif choix=='linéaire quinquonce':
    if angle>=-90 and angle<-26.5:
        xomb1,yomb1,xomb2,yomb2=0,a*l,a*l,a*l/2
    elif angle>=-26.5 and angle<0:
        xomb1,yomb1,xomb2,yomb2=a*l,a*l/2,2*a*l,0
    elif angle>=0 and angle<26.5:
        xomb1,yomb1,xomb2,yomb2=2*a*l,0,a*l,-a*l/2
    elif angle>=26.5 and angle<=90:
        xomb1,yomb1,xomb2,yomb2=a*l,-a*l/2,0,-a*l
```

```
xinter1=(yomb1+xomb1/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ysol/zsol+\
(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)/tan(phi))
yinter1=(xinter1*(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)-xomb1)*1/sin(phi)
xinter2=(yomb2+xomb2/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ysol/zsol+\
(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)/tan(phi))
yinter2=(xinter2*(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)-xomb2)*1/sin(phi)
dico[xinter1],dico[xinter2]=yinter1,yinter2
if (xinter1>l or yinter1>l or yinter1<-l or xinter1<0) and \
(xinter2>l or yinter2>l or yinter2<-l or xinter2<0):
    signal=0
if not(xinter1>l or yinter1>l or yinter1<-l or xinter1<0) and not\
(xinter2>l or yinter2>l or yinter2<-l or xinter2<0):
    signal=2
    ombloc.append(xinter1)
    ombloc.append(xinter2)
    if xinter1>xinter2:
        xh,yh,xb,yb=xinter2,yinter2,xinter1,yinter1
    else:
        xh,yh,xb,yb=xinter1,yinter1,xinter2,yinter2
if (xinter1>l or yinter1>l or yinter1<-l or xinter1<0) and not\
   (xinter2>l or yinter2>l or yinter2<-l or xinter2<0):
    signal=1
    if not(xinter1>l or yinter1>l or yinter1<-l or xinter1<0):
        xinter,yinter=xinter1,yinter1
        ombloc.append(xinter1)
    else:
        xinter,yinter=xinter2,yinter2
        ombloc.append(xinter2)
if angle<-90 or angle>90:
    signal=0
    ombloc=[]
```

```
def ombrage():
# Fonction chargée de calculer les effets d'ombrage
    global eff
    nombreOmbr()
    if signal==0:
        eff=1
    elif signal==1:
        if xinter>=0 and yinter>=0:
            eff=1-(l-xinter)*(l-yinter)/l**2
        else:
            eff=1-(l-xinter)*(l+yinter)/l**2
    else:
        if yh>0 and yb>0 and yh>yb:
            eff=1-((l-yh)*(l-xh)+(yh-yb)*(l-xb))/l**2
        elif yh>0 and yb>0 and yh<yb:
            eff=1-(l-xh)*(l-yh)/l**2
        elif yh<0 and yb<0 and yh<yb:
            eff=1-((l-xh)*(l+yh)+(l-xb)*(yb-yh))/l**2
        elif yh<0 and yb<0 and yh>yb:
            eff=1-(l-xh)*(l+yh)/l**2
        elif yh>0 and yb<0 and yh-yb<l:
            eff=1-((l-xh)*(l+yh)-yh*(l-xb))/l**2
        elif yb<0 and yh>0 and yh-yb>l:
            eff=1-((l-xb)*(l+yb)+(l-xh)*(l-yh))/l**2
        else:
            eff=1-((l-xh)*(l+yh)+(l-xb)*(l-yb))/l**2


# Dernier effet de la disposition linéaire: les blocages.
```

```
def nombreBloc():
# Cette fonction calcul le nombre et la position des obstacles bloquant ainsi
# que les données nécessaires au calcul de l'effet de blocage
    global signal,xh,yh,xb,yb,xinter,yinter,dico
    teta=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    angle=-atan(ytour/xtour)*180/pi
    a=2
    if choix=='linéaire'and variable2==1:
        if angle>=-90 and angle<-45:
            xbloc1,ybloc1,xbloc2,ybloc2=0,2*l,2*l,2*l
        elif angle>=-45 and angle<0:
            xbloc1,ybloc1,xbloc2,ybloc2=2*l,2*l,2*l,0
        elif angle>=0 and angle<45:
            xbloc1,ybloc1,xbloc2,ybloc2=2*l,0,2*l,-2*l
        else:
            xbloc1,ybloc1,xbloc2,ybloc2=2*l,-2*l,0,-2*l
    if choix=='linéaire'and variable2==0:
        if angle>=-90 and angle<-45:
            xbloc1,ybloc1,xbloc2,ybloc2=0,y1,xobst,y2
        elif angle>=-45 and angle<0:
            xbloc1,ybloc1,xbloc2,ybloc2=xobst,y2,xobst,y3
        elif angle>=0 and angle<45:
            xbloc1,ybloc1,xbloc2,ybloc2=xobst,y2,xobst,y3
        else:
            xbloc1,ybloc1,xbloc2,ybloc2=xobst,y3,xobst,y4
    if choix=='linéaire'and variable2==2:
        if angle>=-90 and angle<-45:
            xbloc1,ybloc1,xbloc2,ybloc2=10*l,-l,0,yb2
        elif angle>=-45 and angle<0:
            xbloc1,ybloc1,xbloc2,ybloc2=10*l,-l,10*l,-l
        elif angle>=0 and angle<45:
            xbloc1,ybloc1,xbloc2,ybloc2=10*l,-l,10*l,-l
        else:
            xbloc1,ybloc1,xbloc2,ybloc2=10*l,-l,0,-yb2
```

```
elif choix=='linéaire quinquonce':
    if angle>=-90 and angle<-26.5:
        xbloc1,ybloc1,xbloc2,ybloc2=0,a*l,a*l,a*l/2
    elif angle>=-26.5 and angle<0:
        xbloc1,ybloc1,xbloc2,ybloc2=a*l,a*l/2,2*a*l,0
    elif angle>=0 and angle<=26.5:
        xbloc1,ybloc1,xbloc2,ybloc2=2*a*l,0,a*l,-a*l/2
    elif angle>=26.5 and angle<=90:
        xbloc1,ybloc1,xbloc2,ybloc2=a*l,-a*l/2,0,-a*l
xinter3=(ybloc1+xbloc1/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ytour/htour+\
(cos(teta)*cos(phi)+sin(teta)*xtour/htour)/tan(phi))
yinter3=(xinter3*(cos(teta)*cos(phi)+sin(teta)*xtour/htour)-xbloc1)*1/sin(phi)
xinter4=(ybloc2+xbloc2/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ytour/htour+\
(cos(teta)*cos(phi)+sin(teta)*xtour/htour)/tan(phi))
yinter4=(xinter4*(cos(teta)*cos(phi)+sin(teta)*xtour/htour)-xbloc2)*1/sin(phi)
if jprime==1:
    dico[xinter3],dico[xinter4]=yinter3,yinter4
if (xinter3>l or yinter3>l or yinter3<-l or xinter3<0) and\
(xinter4>l or yinter4>l or yinter4<-l or xinter4<0):
    signal=0
elif not(xinter3>l or yinter3>l or yinter3<-l or xinter3<0) and not\
(xinter4>l or yinter4>l or yinter4<-l or xinter4<0):
    signal=2
    if jprime==1:
        ombloc.append(xinter3)
        ombloc.append(xinter4)
    if xinter3>xinter4:
        xh,yh,xb,yb=xinter4,yinter4,xinter3,yinter3
    else:
        xh,yh,xb,yb=xinter3,yinter3,xinter4,yinter4
else:
    signal=1
```

```
        if not(xinter3>1 or yinter3>1 or yinter3<-1 or xinter3<0):
            xinter,yinter=xinter3,yinter3
            if jprime==1:
                ombloc.append(xinter3)
        else:
            xinter,yinter=xinter4,yinter4
            if jprime==1:
                ombloc.append(xinter4)

def blocage():
# Fonction chargée de calculer les effets d'ombrage
    global eff
    nombreBloc()
    if signal==0:
        eff=1
    elif signal==1:
        if xinter>=0 and yinter>=0:
            eff=1-(l-xinter)*(l-yinter)/l**2
        else:
            eff=1-(l-xinter)*(l+yinter)/l**2
    else:
        if yh>0 and yb>0 and yh>yb:
            eff=1-((l-yh)*(l-xh)+(yh-yb)*(l-xb))/l**2
        elif yh>0 and yb>0 and yh<yb:
            eff=1-(l-xh)*(l-yh)/l**2
        elif yh<0 and yb<0 and yh<yb:
            eff=1-((l-xh)*(l+yh)+(l-xb)*(yb-yh))/l**2
        elif yh<0 and yb<0 and yh>yb:
            eff=1-(l-xh)*(l+yh)/l**2
        elif yh>0 and yb<0 and yh-yb<l:
            eff=1-((l-xh)*(l+yh)-yh*(l-xb))/l**2
        elif yb<0 and yh>0 and yh-yb>l:
            eff=1-((l-xb)*(l+yb)+(l-xh)*(l-yh))/l**2
        else:
            eff=1-((l-xh)*(l+yh)+(l-xb)*(l-yb))/l**2
```

```
# circulaire. Comme signalé précedemment, le calcul de l'effet cosinus est
# indépendant de la disposition. La fonction traceCosCirc peut donc directement
# faire appel aux fonctions préexistantes.

def obstaclecirc():
# Cette fonction détermine la position des obstacles potentiels générant les
# effets d'ombrage dans le cadre d'une configuration circulaire
    global xomb1,yomb1,xomb2,yomb2
    if teta<=90:
        x1=250-(r-2*deltar)*cos((teta)*pi/180)
        y1=sqrt((r-2*deltar)**2-(x1-250)**2)
        x2=250-(r-deltar)*cos((teta-deltateta)*pi/180)
        y2=sqrt((r-deltar)**2-(x2-250)**2)
    else:
        x1=250-(r-2*deltar)*cos(teta*pi/180)
        y1=sqrt((r-2*deltar)**2-(x1-250)**2)
        x2=250-(r-deltar)*cos((teta+deltateta)*pi/180)
        y2=sqrt((r-deltar)**2-(x2-250)**2)
    xomb1,yomb1,xomb2,yomb2=y-y1,x-x1,y-y2,x-x2

def nombrecirc():
# Fonction qui calcul le nombre d'obstacles et les coordonnées d'intersection
    global signal,xh,yh,xb,yb,xinter,yinter
    obstaclecirc()
    teta=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    xinter1=(yomb1+xomb1/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ysol/zsol+\
    (cos(teta)*cos(phi)+sin(teta)*xsol/zsol)/tan(phi))
    yinter1=(xinter1*(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)-xomb1)*1/sin(phi)
    xinter2=(yomb2+xomb2/tan(phi))/(cos(teta)*sin(phi)+sin(teta)*ysol/zsol+\
    (cos(teta)*cos(phi)+sin(teta)*xsol/zsol)/tan(phi))
    yinter2=(xinter2*(cos(teta)*cos(phi)+sin(teta)*xsol/zsol)-xomb2)*1/sin(phi)
    if (xinter1>1 or yinter1>1 or yinter1<-1) and \
    (xinter2>1 or yinter2>1 or yinter2<-1):
        signal=0
```

```
    elif not(xinter1>1 or yinter1>1 or yinter1<-1) and not\
    (xinter2>1 or yinter2>1 or yinter2<-1):
        signal=2
        if xinter1>xinter2:
            xh,yh,xb,yb=xinter2,yinter2,xinter1,yinter1
        else:
            xh,yh,xb,yb=xinter1,yinter1,xinter2,yinter2
    else:
        signal=1
        if not(xinter1>1 or yinter1>1 or yinter1<-1):
            xinter,yinter=xinter1,yinter1
        else:
            xinter,yinter=xinter2,yinter2

def ombrageCirc():
    # Fonction chargée de calculer les effets d'ombrage
    global eff
    nombrecirc()
    if signal==0:
        eff=1
    elif signal==1:
        if xinter>=0 and yinter>=0:
            eff=1-(l-xinter)*(l-yinter)/l**2
        else:
            eff=1-(l-xinter)*(l+yinter)/l**2
    else:
        if yh>0 and yb>0 and yh>yb:
            eff=1-((l-yh)*(l-xh)+(yh-yb)*(l-xb))/l**2
        elif yh>0 and yb>0 and yh<yb:
            eff=1-(l-xh)*(l-yh)/l**2
        elif yh<0 and yb<0 and yh<yb:
            eff=1-((l-xh)*(l+yh)+(l-xb)*(yb-yh))/l**2
        elif yh<0 and yb<0 and yh>yb:
            eff=1-(l-xh)*(l+yh)/l**2
        elif yh>0 and yb<0 and yh-yb<l:
            eff=1-((l-xh)*(l+yh)-yh*(l-xb))/l**2
```

```
        elif yb<0 and yh>0 and yh-yb>1:
            eff=1-((1-xb)*(1+yb)+(1-xh)*(1-yh))/1**2
        else:
            eff=1-((1-xh)*(1+yh)+(1-xb)*(1-yb))/1**2


def blocageCirc():
# Fonction chargée du calcul de l'effet de blocage d'une configuration circulaire
    global eff
    alpha=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    x1=250-(r-deltar)*cos((teta-deltateta)*pi/180)
    y1=sqrt((r-deltar)**2-(x1-250)**2)
    x2=250-(r-deltar)*cos((teta+deltateta)*pi/180)
    y2=sqrt((r-deltar)**2-(x2-250)**2)
    x3=250-(r-2*deltar)*cos(teta*pi/180)
    y3=sqrt((r-2*deltar)**2-(x3-250)**2)
    xbloc1,ybloc1,xbloc2,ybloc2,xbloc3,ybloc3=y-y1,x-x1,y-y2,x-x2,y-y3,x-x3
    xinter1=(ybloc1+xbloc1/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter1=(xinter1*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc1)*1/sin(phi)
    xinter2=(ybloc2+xbloc2/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter2=(xinter2*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc2)*1/sin(phi)
    xinter3=(ybloc3+xbloc3/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter3=(xinter3*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc3)*1/sin(phi)
    if not(xinter1>1 or yinter1>1 or yinter1<-1) and\
    (xinter2>1 or yinter2>1 or yinter2<-1):
        eff=1-(1-xinter1)*(1-yinter1)/1**2
    elif not(xinter2>1 or yinter2>1 or yinter2<-1) and\
    (xinter1>1 or yinter1>1 or yinter1<-1):
        eff=1-(1-xinter2)*(1+yinter2)/1**2
    elif not(xinter1>1 or yinter1>1 or yinter1<-1) and\
    not(xinter2>1 or yinter2>1 or yinter2<-1) and\
    (xinter3>1 or yinter3>1 or yinter3<-1):
```

```
    (xinter3>1 or yinter3>1 or yinter3<-1):
        eff=1-((1-xinter1)*(1-yinter1)+(1-xinter2)*(1+yinter2))/1**2
    elif not (xinter1>1 or yinter1>1 or yinter1<-1) and\
    not(xinter2>1 or yinter2>1 or yinter2<-1) and\
    not(xinter3>1 or yinter3>1 or yinter3<-1):
        eff=1-((1-xinter1)*(1-yinter1)+(1-xinter2)*(1+yinter2)+(1-xinter3)\
        *(1-yinter1+yinter2))/1**2
    else:
        eff=1

def preombloc():
    global xomb1,yomb1,xomb2,yomb2
    if teta<=90:
        x1=250-(r-2*deltar)*cos(teta*pi/180)
        y1=sqrt((r-2*deltar)**2-(x1-250)**2)
        x2=250-(r-deltar)*cos((teta-deltateta)*pi/180)
        y2=sqrt((r-deltar)**2-(x2-250)**2)
    else:
        x1=250-(r-2*deltar)*cos(teta*pi/180)
        y1=sqrt((r-2*deltar)**2-(x1-250)**2)
        x2=250-(r-deltar)*cos((teta+deltateta)*pi/180)
        y2=sqrt((r-deltar)**2-(x2-250)**2)
    xomb1,yomb1,xomb2,yomb2=y-y1,x-x1,y-y2,x-x2

def fombloc1():
    global dico,ombloc
    preombloc()
    dico={}
    ombloc=[]
    alpha=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    xinter1=(yomb1+xomb1/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ysol/zsol+\
    (cos(alpha)*cos(phi)+sin(alpha)*xsol/zsol)/tan(phi))
    yinter1=(xinter1*(cos(alpha)*cos(phi)+sin(alpha)*xsol/zsol)-xomb1)*1/sin(phi)
```

```
    xinter2=(yomb2+xomb2/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ysol/zsol+\
    (cos(alpha)*cos(phi)+sin(alpha)*xsol/zsol)/tan(phi))
    yinter2=(xinter2*(cos(alpha)*cos(phi)+sin(alpha)*xsol/zsol)-xomb2)*1/sin(phi)
    dico[xinter1],dico[xinter2]=yinter1,yinter2
    if not(xinter1>1 or yinter1>1 or yinter1<-1) and not\
    (xinter2>1 or yinter2>1 or yinter2<-1):
        ombloc.append(xinter1)
        ombloc.append(xinter2)
    elif not(xinter1>1 or yinter1>1 or yinter1<-1) and\
    (xinter2>1 or yinter2>1 or yinter2<-1):
        ombloc.append(xinter1)
    elif not(xinter2>1 or yinter2>1 or yinter2<-1) and\
    (xinter1>1 or yinter1>1 or yinter1<-1):
        ombloc.append(xinter2)

def fombloc2():
    fombloc1()
    alpha=atan(sqrt(xn**2+yn**2)/zn)
    phi=atan(yn/xn)
    x1=250-(r-deltar)*cos((teta-deltateta)*pi/180)
    y1=sqrt((r-deltar)**2-(x1-250)**2)
    x2=250-(r-deltar)*cos((teta+deltateta)*pi/180)
    y2=sqrt((r-deltar)**2-(x2-250)**2)
    x3=250-(r-2*deltar)*cos(teta*pi/180)
    y3=sqrt((r-2*deltar)**2-(x3-250)**2)
    xbloc1,ybloc1,xbloc2,ybloc2,xbloc3,ybloc3=y-y1,x-x1,y-y2,x-x2,y-y3,x-x3
    xinter3=(ybloc1+xbloc1/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter3=(xinter3*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc1)*1/sin(phi)
    xinter4=(ybloc2+xbloc2/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter4=(xinter4*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc2)*1/sin(phi)
    xinter5=(ybloc3+xbloc3/tan(phi))/(cos(alpha)*sin(phi)+sin(alpha)*ytour/htour+\
```

```
    (cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)/tan(phi))
    yinter5=(xinter5*(cos(alpha)*cos(phi)+sin(alpha)*xtour/htour)-xbloc3)*1/sin(phi)
    dico[xinter3],dico[xinter4],dico[xinter5]=yinter3,yinter4,yinter5
    if not(xinter3>1 or yinter3>1 or yinter3<-1) and\
    (xinter4>1 or yinter4>1 or yinter4<-1):
        ombloc.append(xinter3)
    elif not(xinter4>1 or yinter4>1 or yinter4<-1) and\
    (xinter3>1 or yinter3>1 or yinter3<-1):
        ombloc.append(xinter4)
    elif not(xinter3>1 or yinter3>1 or yinter3<-1) and\
    not(xinter4>1 or yinter4>1 or yinter4<-1) and\
    (xinter5>1 or yinter5>1 or yinter5<-1):
        ombloc.append(xinter3)
        ombloc.append(xinter4)
    elif not (xinter3>1 or yinter3>1 or yinter3<-1) and\
    not(xinter4>1 or yinter4>1 or yinter4<-1) and\
    not(xinter5>1 or yinter5>1 or yinter5<-1):
        ombloc.append(xinter3)
        ombloc.append(xinter4)
        ombloc.append(xinter5)

def calculOmbloc():
    global eff
    if choix=='circulaire':
        fombloc2()
    elif choix=='linéaire' or choix=='linéaire quinquonce':
        nombreOmbr()
        nombreBloc()
    ombloc1=[]
    ombloc2=[]
    if ombloc==[]:
        eff=1
```

```
elif len(ombloc)==1:
    if dico[ombloc[0]]<=0:
        eff=1-(l-ombloc[0])*(l+dico[ombloc[0]])/l**2
    else:
        eff=1-(l-ombloc[0])*(l-dico[ombloc[0]])/l**2
else:
    ombloc.sort()
    for i in range(len(ombloc)):
        ombloc1.append(dico[ombloc[i]])
    for i in range(len(ombloc)):
        ombloc[i]=l-ombloc[i]
        if ombloc1[i]<=0:
            ombloc2.append(l+ombloc1[i])
            ombloc1[i]=0
        else:
            ombloc2.append(l)
    second=[[]]*len(ombloc)
    for i in range(len(ombloc)):
        second[i]=[0]*int(ombloc1[i]*100)+[1]*(int(ombloc2[i]*100)\
        -int(ombloc1[i]*100))+[0]*(int(l)*100-int(ombloc2[i]*100))
    som=[]
    x,z,aire=0,0,0
    for k in range(len(second[0])):
        z=z+second[0][k]
    for i in range(1,len(second)):
        x=0
        for k in range(len(second[0])):
            if (second[0][k]-second[i][k])==-1:
                second[0][k]=1
                x+=1
        som.append(x)
    for i in range(len(som)):
        aire=aire+float(som[i])/100*ombloc[i+1]
    eff=1-(aire+float(z)/100*ombloc[0])/l**2
```

```
        eff=1-(aire+float(z)/100*ombloc[0])/l**2

def position2():
    global azimu,alt,h,l,deb,fi,moi
    num=275*eval(mois.get())/9-2*(eval(mois.get())+9)/12+eval(jour.get())-30
    horaireloc=heur+minut/60.+eval(seconde.get())/3600
    h,l,deb,fi,ref,moi=eval(hauteur.get()),eval(taille.get()),eval(debut.get()),\
    eval(fin.get()),eval(reflexion.get()),eval(mois.get())
    TSM=horaireloc-1.558/15
    positionabs=360*(num-1)/365.242
    eot=0.258*cos(positionabs*pi/180)-7.416*sin(positionabs*pi/180)-3.648*\
    cos(2*positionabs*pi/180)-9.228*sin(2*positionabs*pi/180)
    TSV=TSM+eot/60
    declinaison=(asin(0.39795*cos(0.98563*(num-173)*pi/180)))
    w=(TSV-12)*15
    alt=(asin(sin(declinaison)*sin(12.459*pi/180)+cos(declinaison)*\
    cos(w*pi/180)*cos(12.459*pi/180)))*180/pi
    azimutn=(acos((sin(declinaison)*cos(12.459*pi/180)-cos(declinaison)*\
    cos(w*pi/180)*sin(12.459*pi/180))/cos(alt*pi/180)))*180/pi
    if sin(w*pi/180)>0:
        azimu=180-azimutn
    else:
        azimu=-180+azimutn
```

```
###############################################################################
#                            PROGRAMME PRINCIPAL                              #
# Nous allons à présent utiliser les fonctions définies ci dessus pour        #
# effectuer des opérations sur les données récupérées à l'interface graphique. #
# Nous pourrons ensuite retranscrire les résultats sous forme numérique ou sous#
# forme graphique au niveau de l'interface.                                   #
###############################################################################



def trace(f,i,j,k):
    global x,y,r,teta,deltar,deltateta,eff,jprime
    jprime=0
    if j==1 or k==1:
        jprime=1
    if choix=='linéaire':
        canevas.delete(ALL)
        y=0.01
        while y<500:
            x=0.01
            while x<500:
                effetCos()
                eff1=eff
                if f==1:
                    ombrage()
                elif i==1:
                    blocage()
                elif j==1:
                    calculOmbloc()
                elif k==1:
                    calculOmbloc()
                    eff=eff*eff1*ref
                couleur()
                canevas.create_rectangle(x,y,x+2,y+2,outline=c,fill=c)
```

```
            x+=2
        y+=2
elif choix=='circulaire':
    r,deltar,deltateta,n,teta=57.296,2,0.73,1,40
    canevas.delete(ALL)
    while r<370:
        if n%2==0:
            teta=teta+2*deltateta
        else:
            teta=teta+deltateta
        while teta<140:
            x=250-r*cos(teta*pi/180)
            y=sqrt(r**2-(x-250)**2)
            effetCos()
            eff1=eff
            if f==1:
                ombrageCirc()
            elif i==1:
                blocageCirc()
            elif j==1:
                calculOmbloc()
            elif k==1:
                calculOmbloc()
                eff=eff*eff1*ref
            couleur()
            canevas.create_rectangle(x-2,y-2,x+2,y+2,outline=c,fill=c)
            teta=teta+2*deltateta
        teta=40
        n+=1
        r+=deltar
elif choix=='linéaire quinquonce':
```

```
        n=1
        y=0.01
        while y<500:
            if n%2==0:
                x=0.01
            else:
                x=l/2+0.01
            while x<500:
                effetCos()
                eff1=eff
                if f==1:
                    ombrage()
                elif i==1:
                    blocage()
                elif j==1:
                    calculOmbloc()
                elif k==1:
                    calculOmbloc()
                    eff=eff*eff1*ref
                couleur()
                canevas.create_rectangle(x,y,x+2,y+2,outline=c,fill=c)
                x+=1.2*l
            y+=1.2*l
            n+=1


def dispoType(event):
    canevas2.delete(ALL)
    canevas2.create_line(230,3,270,3,width=5,fill='#000099')
    canevas2.create_line(250,0,250,20,width=5,fill='#000099')
    if choix=='circulaire':
        r,deltar,deltateta,n,teta=57.296,10,8,1,40
```

```
    r,deltar,deltateta,n,teta=57.296,10,8,1,40
    while r<370:
        if n%2==0:
            teta=teta+2*deltateta
        else:
            teta=teta+deltateta
        while teta<140:
            x=250-r*cos(teta*pi/180)
            y=sqrt(r**2-(x-250)**2)
            canevas2.create_rectangle(x-4,y-4,x+4,y+4,outline='blue')
            teta=teta+2*deltateta
        teta=40
        n+=1
        r+=deltar
elif choix=='linéaire':
    y=30
    while y<450:
        x=50
        while x<450:
            canevas2.create_rectangle(x-4,y-4,x+4,y+4,outline='blue')
            x+=30
        y+=30
elif choix=='linéaire quinquonce':
    n=1
    y=50
    while y<450:
        if n%2==0:
            x=50+15
        else:
            x=50
        while x<450:
            canevas2.create_rectangle(x-4,y-4,x+4,y+4,outline='blue')
            x+=30
        n+=1
        y+=30
```

```
def effaceGraf2(event):
    canevas2.delete(ALL)
def effaceGraf1(event):
    canevas.delete(ALL)

def ressource(event):
    canevas2.create_line(40,400,40,100,arrow=LAST)
    canevas2.create_line(40,400,475,400,arrow=LAST)
    canevas2.create_text(40,420,text='7H30',fill='#999999')
    canevas2.create_text(240,420,text='12H30',fill='#999999')
    canevas2.create_text(400,420,text='16H30',fill='#999999')
    canevas2.create_text(20,375,text='100',fill='#999999')
    canevas2.create_text(20,275,text='500',fill='#999999')
    canevas2.create_text(20,150,text='1000',fill='#999999')
    canevas2.create_text(460,420,text='t(heures)',fill='#999999')
    canevas2.create_text(40,90,text='P(W/m2)',fill='#999999')
    canevas2.create_text(120,440,text='Janvier',fill=coul[0])
    canevas2.create_text(120,460,text='Février',fill=coul[15])
    canevas2.create_text(120,480,text='Mars',fill=coul[30])
    canevas2.create_rectangle(90,437,96,443,fill=coul[0],outline=coul[0])
    canevas2.create_rectangle(90,457,96,463,fill=coul[15],outline=coul[15])
    canevas2.create_rectangle(90,477,96,483,fill=coul[30],outline=coul[30])
    canevas2.create_rectangle(170,437,176,443,fill=coul[45],outline=coul[45])
    canevas2.create_rectangle(170,457,176,463,fill=coul[60],outline=coul[60])
    canevas2.create_rectangle(170,477,176,483,fill=coul[75],outline=coul[75])
    canevas2.create_text(200,440,text='Avril',fill=coul[45])
    canevas2.create_text(200,460,text='Mai',fill=coul[60])
    canevas2.create_text(200,480,text='Juin',fill=coul[75])
    canevas2.create_rectangle(250,437,256,443,fill=coul[90],outline=coul[90])
    canevas2.create_rectangle(250,457,256,463,fill=coul[105],outline=coul[105])
    canevas2.create_rectangle(250,477,256,483,fill=coul[120],outline=coul[120])
    canevas2.create_text(290,440,text='Juillet',fill=coul[90])
```

```
    canevas2.create_text(290,460,text='Aôut',fill=coul[105])
    canevas2.create_text(290,480,text='Septembre',fill=coul[120])
    canevas2.create_rectangle(340,437,346,443,fill=coul[135],outline=coul[135])
    canevas2.create_rectangle(340,457,346,463,fill='#c00000',outline='#c00000')
    canevas2.create_rectangle(340,477,346,483,fill='purple',outline='purple')
    canevas2.create_text(380,440,text='Octobre',fill=coul[135])
    canevas2.create_text(380,460,text='Novembre',fill='#c00000')
    canevas2.create_text(380,480,text='Décembre',fill='purple')
    for i in range(2,12):
        canevas2.create_line(i*40,400,i*40,125,fill='#bbbbbb')
    for i in range(5,16):
        canevas2.create_line(40,i*25,440,i*25,fill='#bbbbbb')
    if int(moi)<=10:
        a=coul[15*(int(moi)-1)]
    elif int(moi)==11:
        a='#c00000'
    else:
        a='purple'
    for i in range(1,57):
        canevas2.create_line(40+i*40./6,400-annee[int(moi)][i]/100*25,
        40+(i+1)*40./6,400-annee[int(moi)][i+1]/100*25,fill=a,width=2)

def evolPuiss(event):
    global x,y,teta,r,deltar,deltateta,variable,heur,minut,jprime
    variable=1
    if int(moi)<=10:
        a=coul[15*(int(moi)-1)]
    elif int(moi)==11:
        a='#c00000'
    else:
        a='purple'
    if choix=='circulaire':
```

```
l2=[]
heur,minut=7,30
for i in range(1,57):
    if minut==50:
        heur=heur+1
        minut=0
    else:
        minut+=10
    r,deltar,deltateta,n,teta=57.296,10,8,1,40
    l1=[]
    while r<370:
        if n%2==0:
            teta=teta+2*deltateta
        else:
            teta=teta+deltateta
        while teta<140:
            x=250-r*cos(teta*pi/180)
            y=sqrt(r**2-(x-250)**2)
            position2()
            effetCos()
            eff1=eff
            calculOmbloc()
            p=eff1*eff*ref*annee[int(moi)][i]/1000*0.8**2
            l1.append(p)
            teta+=2*deltateta
        teta=40
        n+=1
        r+=deltar
    b=0
    for t in range(len(l1)):
        b+=l1[t]
    l2.append(b)
for i in range(len(l2)-1):
```

```
            canevas2.create_line(40+i*40./6,400-l2[i]/10*25,
            40+(i+1)*40./6,400-l2[i+1]/10*25,fill=a,width=2)
    elif choix=='linéaire':
        jprime=1
        l2=[]
        heur,minut=7,30
        for i in range(1,57):
            if minut==50:
                heur=heur+1
                minut=0
            else:
                minut+=10
            l1=[]
            y=50
            while y<450:
                x=50
                while x<450:
                    position2()
                    effetCos()
                    eff1=eff
                    calculOmbloc()
                    p=eff1*eff*ref*annee[int(moi)][i]/1000*0.8**2
                    l1.append(p)
                    x+=30
                y+=30
            b=0
            for t in range(len(l1)):
                b+=l1[t]
            l2.append(b)
        for i in range(len(l2)-1):
            canevas2.create_line(40+i*40./6,400-l2[i]/10*25,
            40+(i+1)*40./6,400-l2[i+1]/10*25,fill=a,width=2)
```

```
elif choix=='linéaire quinquonce':
    jprime=1
    l2=[]
    heur,minut=7,30
    for i in range(1,57):
        if minut==50:
            heur=heur+1
            minut=0
        else:
            minut+=10
        l1=[]
        n=1
        y=50
        while y<450:
            if n%2==0:
                x=50+15
            else:
                x=50
            while x<450:
                position2()
                effetCos()
                eff1=eff
                calculOmbloc()
                p=eff1*eff*ref*annee[int(moi)][i]/1000*0.8**2
                l1.append(p)
                x+=30
            n+=1
            y+=30
        b=0
        for t in range(len(l1)):
            b+=l1[t]
        l2.append(b)
    for i in range(len(l2)-1):
        canevas2.create_line(40+i*40./6,400-l2[i]/10*25,
        40+(i+1)*40./6,400-l2[i+1]/10*25,fill=a,width=2)
```

```
canevas2.create_line(40,400,40,100,arrow=LAST)
canevas2.create_line(40,400,475,400,arrow=LAST)
canevas2.create_text(40,420,text='7H30',fill='#999999')
canevas2.create_text(240,420,text='12H30',fill='#999999')
canevas2.create_text(400,420,text='16H30',fill='#999999')
canevas2.create_text(20,375,text='10',fill='#999999')
canevas2.create_text(20,275,text='50',fill='#999999')
canevas2.create_text(20,150,text='100',fill='#999999')
canevas2.create_text(460,420,text='t(heures)',fill='#999999')
canevas2.create_text(40,90,text='P(kW)',fill='#999999')
canevas2.create_text(120,440,text='Janvier',fill=coul[0])
canevas2.create_text(120,460,text='Février',fill=coul[15])
canevas2.create_text(120,480,text='Mars',fill=coul[30])
canevas2.create_rectangle(90,437,96,443,fill=coul[0],outline=coul[0])
canevas2.create_rectangle(90,457,96,463,fill=coul[15],outline=coul[15])
canevas2.create_rectangle(90,477,96,483,fill=coul[30],outline=coul[30])
canevas2.create_rectangle(170,437,176,443,fill=coul[45],outline=coul[45])
canevas2.create_rectangle(170,457,176,463,fill=coul[60],outline=coul[60])
canevas2.create_rectangle(170,477,176,483,fill=coul[75],outline=coul[75])
canevas2.create_text(200,440,text='Avril',fill=coul[45])
canevas2.create_text(200,460,text='Mai',fill=coul[60])
canevas2.create_text(200,480,text='Juin',fill=coul[75])
canevas2.create_rectangle(250,437,256,443,fill=coul[90],outline=coul[90])
canevas2.create_rectangle(250,457,256,463,fill=coul[105],outline=coul[105])
canevas2.create_rectangle(250,477,256,483,fill=coul[120],outline=coul[120])
canevas2.create_text(290,440,text='Juillet',fill=coul[90])
canevas2.create_text(290,460,text='Août',fill=coul[105])
canevas2.create_text(290,480,text='Septembre',fill=coul[120])
canevas2.create_rectangle(340,437,346,443,fill=coul[135],outline=coul[135])
canevas2.create_rectangle(340,457,346,463,fill='#c00000',outline='#c00000')
canevas2.create_rectangle(340,477,346,483,fill='purple',outline='purple')
canevas2.create_text(380,440,text='Octobre',fill=coul[135])
```

```
    canevas2.create_text(380,460,text='Novembre',fill='#c00000')
    canevas2.create_text(380,480,text='Décembre',fill='purple')
    for i in range(2,12):
        canevas2.create_line(i*40,400,i*40,125,fill='#bbbbbb')
    for i in range(5,16):
        canevas2.create_line(40,i*25,440,i*25,fill='#bbbbbb')



##############################################################################
#                          Algorithme d'optimisation                         #
##############################################################################

def choix_ligne1():
    global x,y,jprime,ydeb,variable2,ystoc
    ystoc=[]
    variable2=1
    jprime=1
    stoc=0
    for i in range(10,50):
        y=i
        som=0
        while y<500+i:
            x=1
            while x<499:
                effetCos()
                eff1=eff
                calculOmbloc()
                etot=eff*eff1*ref
                som=som+etot
                x+=2*l
            y+=2*l
        if stoc<=som:
            stoc=som
            ydeb=i
            ystoc.append(ydeb)
```

```
def ligne1():
    global x,y,lstoc,yb2,variable2,p
    lstoc=[[]]*50
    variable2=2
    x,y,emin,p,d,em=250-1.4*l,ydeb,0.75,0,25,0
    lstoc[1].append(250)
    while x>10:
        effetCos()
        eff1=eff
        yb2=lstoc[1][len(lstoc[1])-1]-x
        nombreBloc()
        blocage()
        etot=eff*eff1*ref
        if etot>emin:
            lstoc[1].append(x)
            lstoc[1].append(x+2*(250-x))
            p+=2*etot*annee[int(moi)][27]/1000.*(l/10.)**2
            x=x-1.4*l
        else:
            x=x-2
    lstoc[1].sort()


def l_deb():
    global x,y,y1,y2,y3,y4,xobst,k,lstoc,ydeb,variable2,yd
    variable2=0
    emin=0.77
    lstoc[m]=[]
    for i in range(len(lstoc[m-1])-1):
        lstoc[m].append(lstoc[m-1][i]+(lstoc[m-1][i+1]-lstoc[m-1][i])/2)
    y=ydeb+1.2*l
```

```
while y<ydeb+1.2*l+100:
    som=0
    for k in range(len(lstoc[m])):
        x=lstoc[m][k]
        for i in range(len(lstoc[m-1])):
            if lstoc[m-1][i]>x and lstoc[m-1][i-1]<x:
                y2,y3,xobst=x-lstoc[m-1][i-1],x-lstoc[m-1][i],y-ydeb
        if k!=0 and k!=len(lstoc[m])-1:
            y1,y4=x-lstoc[m][k-1],x-lstoc[m][k+1]
        elif k==len(lstoc[m])-1:
            y1,y4=x-lstoc[m][k-1],-10*l
        else:
            y1,y4=10*l,x-lstoc[m][k+1]
        effetCos()
        eff1=eff
        calculOmbloc()
        etot=eff*eff1*ref
        som=som+etot
    moy=som/len(lstoc[m])
    if moy>=emin:
        yd=ydeb
        ydeb=y
        ystoc.append(ydeb)
        break
    else:
        y+=2
```

```
def l_fin():
    emin=0.7
    l_deb()
    global x,y,y1,y2,y3,y4,xobst,ydeb,lstoc,m,put,p
    if lstoc[m-1][0]>15:
        x,y=lstoc[m-1][0]-l,ydeb
    else:
        x,y=lstoc[m-1][0]+l,ydeb
    lstoc[m]=[]
    while x<490:
        for j in range(30):
            x=x+j
            for i in range(len(lstoc[m-1])):
                if lstoc[m-1][i]>x and lstoc[m-1][i-1]<x:
                    y2,y3,xobst=x-lstoc[m-1][i-1],x-lstoc[m-1][i],ydeb-yd
            if lstoc[m]!=[] and x<485:
                y1,y4=x-lstoc[m][len(lstoc[m])-1],lstoc[m][len(lstoc[m])-1]-x
            if lstoc[m]==[]:
                y1,y4=10*l,lstoc[m-1][1]-lstoc[m-1][0]
            if x>489:
                y1,y4=x-lstoc[m][len(lstoc[m])-1],-10*l
            effetCos()
            eff1=eff
            calculOmbloc()
            etot=eff*eff1*ref
            if etot>emin:
                lstoc[m].append(x)
                p+=etot*annee[int(moi)][27]/1000.*(l/10.)**2
                break
            x=x-j
        x+=1.4*l
    put=p
```

```
def optimise():
    global m,place,placement
    place,placement=[],[]
    choix_ligne1()
    ligne1()
    for m in range(2,50):
        l_fin()
        if put>=300:
            break
    a=0
    for k in range(1,m+1):
        a+=len(lstoc[k])
        place.append(lstoc[k])
        placement.append([])


def tracedispo(event):
    global x,y,etot
    optimise()
    for k in range(len(place)):
        for i in range(len(place[k])):
            if k==0:
                placement[k].append(place[k][i])
            if k!=0 and place[k][i]>250 and place[k][i-1]<250:
                if place[k][i]-250<=l/2:
                    placement[k].append(250)
                if place[k][i]-250>l/2:
                    placement[k].append(place[k][i])
                    placement[k].append(250-(place[k][i]-250))
            if k!=0 and place[k][i]>250 and place[k][i-1]==250:
                placement[k].append(250)
                placement[k].append(place[k][i])
                placement[k].append(250-(place[k][i]-250))
```

```
        if k!=0 and place[k][i]>250 and place[k][i-1]>250:
            placement[k].append(place[k][i])
            placement[k].append(250-(place[k][i]-250))
p,d,em,n=0,0,0,0
for k in range(len(placement)):
    placement[k].sort()
for k in range(len(placement)):
    y=ystoc[k]
    for i in range(len(placement[k])):
        x=placement[k][i]
        if i==0:
            y1,y4=10*l,placement[k][1]-placement[k][0]
            if placement[k-1][0]>x:
                y2,y3,xobst=10*l,x-placement[k-1][0],ystoc[k]-ystoc[k-1]
            else:
                y2,y3,xobst=x-placement[k-1][0],x-placement[k-1][1],ystoc[k]-\
                            ystoc[k-1]
        if i==len(placement[k])-1:
            y1,y4=x-placement[k][i-1],-10*l
            if placement[k-1][len(placement[k-1])-1]>x:
                y2,y3,xobst=x-placement[k-1][len(placement[k-1])-2],x-placement[k-1]\
                            [len(placement[k-1])-1],
                            ystoc[k]-ystoc[k-1]
            else:
                y2,y3,xobst=x-placement[k-1][len(placement[k-1])-1],-10*l,ystoc[k]-\
                            ystoc[k-1]
        else:
            y1,y4=x-placement[k][i-1],x-placement[k][i+1]
            for m in range(len(placement[k-1])):
                if placement[k-1][m]>x and placement[k-1][m-1]<x:
                    y2,y3,xobst=x-placement[k-1][m-1],x-placement[k-1][m],ystoc[k]-\
                                ystoc[k-1]
```

```
            effetCos()
            eff1=eff
            calculOmbloc()
            etot=eff1*eff*ref
            couleur()
            n+=1
            canevas.create_rectangle(x-l/2,ystoc[k]-l/2,x+l/2,ystoc[k]+l/2,
                                     fill=c)
            p+=etot*(l/10.)**2*annee[int(moi)][27]/1000.
            d+=sqrt(((250-x)/10.)**2+(y/10.)**2+h**2)
            em+=etot
    dut=d/n
    emut=em/n
    etiquette1.configure(text=str(n)+' kW')
    etiquette2.configure(text=str(int(dut*100)/100.)+' m')
    etiquette3.configure(text=str(int(emut*100)/100.))


def rec_puiss():
    global x,y,puiss,heur,minut,variable,y1,y2,y3,y4,xobst
    heur,minut,puiss,variable=7.,30.,[],1
    for m in range(1,57):
        p=0
        if minut==50:
            heur=heur+1
            minut=0
        else:
            minut+=10
        for k in range(len(placement)):
            y=ystoc[k]
            for i in range(len(placement[k])):
```

```
                x=placement[k][i]
                position2()
                effetCos()
                eff1=eff
                calculOmbloc()
                p+=eff1*eff*ref*annee[int(moi)][m]/1000*(1/10.)**2
        puiss.append(p)

def trace_puiss(event):
    rec_puiss()
    canevas2.create_line(40,400,40,100,arrow=LAST)
    canevas2.create_line(40,400,475,400,arrow=LAST)
    canevas2.create_text(40,420,text='7H30',fill='#999999')
    canevas2.create_text(240,420,text='12H30',fill='#999999')
    canevas2.create_text(400,420,text='16H30',fill='#999999')
    canevas2.create_text(20,375,text='30',fill='#999999')
    canevas2.create_text(20,275,text='150',fill='#999999')
    canevas2.create_text(20,150,text='300',fill='#999999')
    canevas2.create_text(460,420,text='t(heures)',fill='#999999')
    canevas2.create_text(40,90,text='P(kW)',fill='#999999')
    canevas2.create_text(120,440,text='Janvier',fill=coul[0])
    canevas2.create_text(120,460,text='Février',fill=coul[15])
    canevas2.create_text(120,480,text='Mars',fill=coul[30])
    canevas2.create_rectangle(90,437,96,443,fill=coul[0],outline=coul[0])
    canevas2.create_rectangle(90,457,96,463,fill=coul[15],outline=coul[15])
    canevas2.create_rectangle(90,477,96,483,fill=coul[30],outline=coul[30])
    canevas2.create_rectangle(170,437,176,443,fill=coul[45],outline=coul[45])
    canevas2.create_rectangle(170,457,176,463,fill=coul[60],outline=coul[60])
    canevas2.create_rectangle(170,477,176,483,fill=coul[75],outline=coul[75])
    canevas2.create_text(200,440,text='Avril',fill=coul[45])
    canevas2.create_text(200,460,text='Mai',fill=coul[60])
    canevas2.create_text(200,480,text='Juin',fill=coul[75])
```

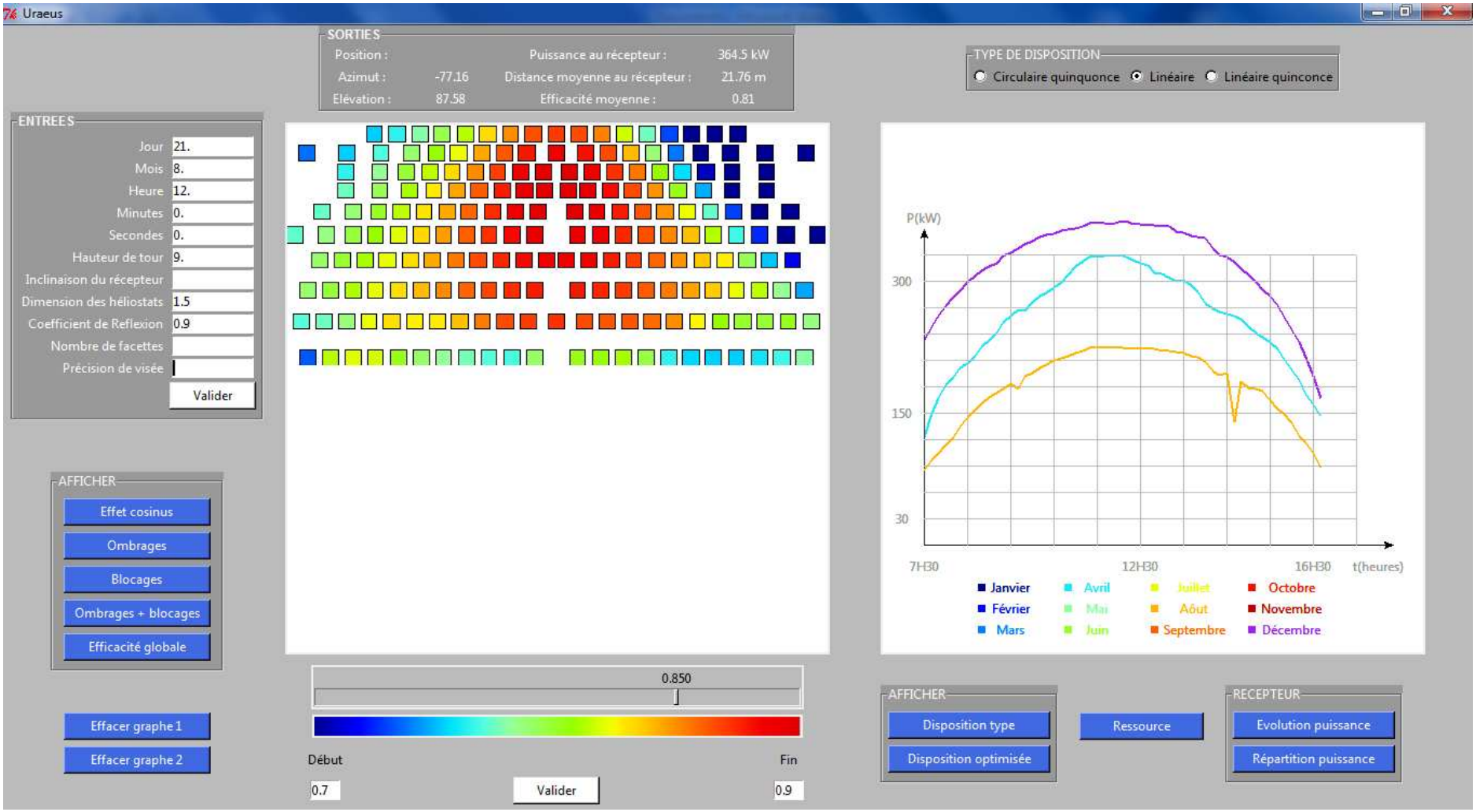
```
canevas2.create_rectangle(250,437,256,443,fill=coul[90],outline=coul[90])
canevas2.create_rectangle(250,457,256,463,fill=coul[105],outline=coul[105])
canevas2.create_rectangle(250,477,256,483,fill=coul[120],outline=coul[120])
canevas2.create_text(290,440,text='Juillet',fill=coul[90])
canevas2.create_text(290,460,text='Aôut',fill=coul[105])
canevas2.create_text(290,480,text='Septembre',fill=coul[120])
canevas2.create_rectangle(340,437,346,443,fill=coul[135],outline=coul[135])
canevas2.create_rectangle(340,457,346,463,fill='#c00000',outline='#c00000')
canevas2.create_rectangle(340,477,346,483,fill='purple',outline='purple')
canevas2.create_text(380,440,text='Octobre',fill=coul[135])
canevas2.create_text(380,460,text='Novembre',fill='#c00000')
canevas2.create_text(380,480,text='Décembre',fill='purple')
for i in range(2,12):
    canevas2.create_line(i*40,400,i*40,125,fill='#bbbbbb')
for i in range(5,16):
    canevas2.create_line(40,i*25,440,i*25,fill='#bbbbbb')
if int(moi)<=10:
    a=coul[15*(int(moi)-1)]
elif int(moi)==11:
    a='#c00000'
else:
    a='purple'
for i in range(len(puiss)-1):
    canevas2.create_line(40+i*40./6,400-puiss[i]/30*25,
        40+(i+1)*40./6,400-puiss[i+1]/30*25,fill=a,width=2)
```

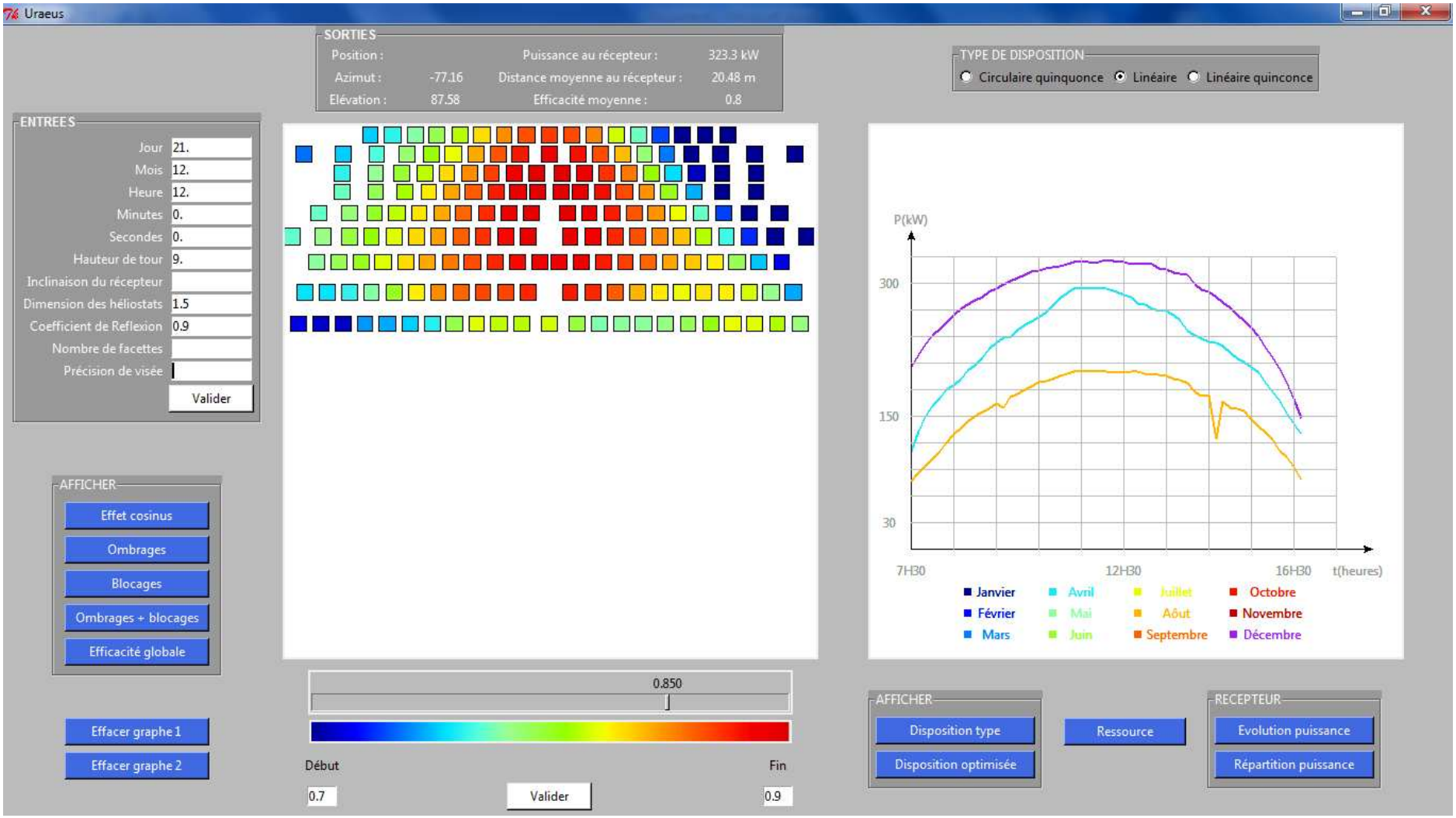
```
# Liaison de la zone graphique à certaines fonctions
canevas.bind('<Button-1>',coordonnees)
b1.bind('<Button-1>',position)
b12.bind('<Button-1>',position)
b8.bind('<Button-1>',ressource)
b10.bind('<Button-1>',evolPuiss)
b72.bind('<Button-1>',effaceGraf2)
b7.bind('<Button-1>',effaceGraf1)
b13.bind('<Button-1>',trace_puiss)
b9.bind('<Button-1>',tracedispo)


annee=[[]]*13

annee[1]=[410.9,492.92,543,585.9,623.6,651,679,702,720.8,737.9,758.2,767.7,777.7,
        791.4,806.2,814.2,826.3,833.9,837.2,851.9,857,865.3,874.7,882.8,893.4,
        893,894,890.6,887.9,901.3,891.4,884,878.8,882.2,873.2,884.1,875,864.3,863.4,
        843.3,850,829.2,811.7,807,807,786.6,774.9,760,750.8,743.4,715.8,684.1,
        657.6,626.8,584.7,535.4,467.1,398.2]

annee[2]=[243.3,330.7,353.7,393.2,417.7,410.8,523.1,560.4,596.8,551.8,550.5,663,
         672.4,673,607,647.6,725.6,739.9,749,749.5,748.4,754,755.7,754.7,771,
         780.3,774,783,788.4,788.2,787.8,781.1,754.5,770.3,774.2,746.5,726.9,
         735.5,731.5,749.2,719.7,719.6,725.7,707.4,707.8,696.4,688.7,659.5,641.6,
         576.5,565,561.6,546.7,517.9,452.6,400,363.6,315.5]

annee[3]=[196,233.6,294.2,335.6,368.4,399,428.7,469.9,498.9,515.1,555.3,577.7,590.6,
      608.4,610.4,637,650,655,650.3,673.3,674.7,684.9,705.5,711.3,705.7,708.1,
      717.6,719.5,725.9,713.8,723.1,725,723.2,724.8,717,706.6,700.3,693.6,682.7,680.5,
      680.8,668.5,646,640.4,631.2,610.3,592.9,533,556.1,540.6,507.4,489.4,467.1,
      316.7,398.4,354.3,156,17]
```

```
annee[4]=[369.2,423.7,508.1,558.3,602.2,624.9,652.6,665.4,685,712,728,747.7,778,
       796.6,809.4,803.4,824.6,840.8,847.6,861.4,873.7,896.3,918,934.6,946.3,
       943.9,944.2,943,945.1,936,926.7,915.3,908,885.1,881.4,870,862.1,867.3,851.7,
       836,801.6,789.1,781.9,777.8,778.8,773.7,754.3,736.9,731.9,718.6,704.7,
       675.3,641.9,612.2,570.1,535.9,497.8,447]

annee[5]=[257.1,330.9,371.3,399.5,434,454.6,480.9,505.9,523.2,541.8,559.6,578.2,
     589.2,604.4,611.8,619.6,625.7,613.9,620.5,626.8,640.1,631.4,637,635.2,
     641.6,645.4,644.6,650,651.7,657,653.2,650.6,652.5,646.5,647.6,647.7,651.2,
     644.9,655.3,643.3,658.5,629.6,609.4,606.9,600.7,578.2,539,542.4,526.8,
     493.2,471.2,466.9,434.8,400.5,356.7,326.8,217.5,260.4]

annee[6]=[492.4,486,539,542,567.8,598.2,602.1,620.6,648,667.8,680.5,693,696.5,701.3,
      692,703.9,711.9,749.1,755.7,774,784,782.3,771.2,768,804.8,804.4,806.9,
      809.3,807.2,807.3,809.2,803.4,793.7,799,792.6,790.8,789.1,798.7,782.6,786.1,
      783.5,778.8,774,765.7,758.2,744.3,745.1,733.3,715.8,698.7,672.4,647.2,
      624.4,598.5,576.8,491.6,461.9,457.5]

annee[7]=[395.5,431.6,456.2,485.8,520.9,567.1,561.5,579.7,601.4,620.6,636.4,661,
         668.1,681.2,684.8,694.4,700,702.4,700.9,705.1,701.6,703.3,702,696.3,705.2,
         708.5,715,712.1,715.4,716.8,715.8,709.6,704.4,700.2,692.6,688.6,685,
         678.1,672.4,664,662.3,651.6,639.6,633,621.4,616.6,606.6,595.4,575.1,
         561.2,541.6,519.8,500.6,471.7,439.6,415.6,386,330]

annee[8]=[263,297,332,354.8,379.7,405.6,439.4,467.7,490.8,508.4,528.4,539.8,549.4,
      563.2,543,580.4,589,602.4,607.9,621.2,623.6,627.1,635.6,641.5,648.4,647.5,
      650,645.6,646.3,640,638.1,639.2,641.3,640.8,633.9,632.2,633.2,629.4,622.6,
      618.6,609.3,580.8,570.1,578.8,414.3,560.2,542.3,543.3,540.6,512,488.6,472,
      446.8,402.2,384.8,347.7,296.3,247.4]
```

```
annee[9]=[396.8,435.9,466.2,492.1,517,535,553.9,573.3,591.1,604.5,618.3,633.6,
           646.2,659.4,672,683.8,693.7,701.2,709.2,714.8,719.2,719.6,723.2,728.9,
           728.3,732.1,730,731.8,731.9,732.3,733.9,735.5,734,728.4,725.8,722.5,
           718.3,713.9,703.1,699.7,689.5,681.2,629.7,667.4,656.7,634.9,651.8,
           605.6,630.1,557,542.4,498,432,437,431.9,394.5,334.1,297.6]

annee[10]=[332,389.9,420.2,477.1,524.6,555,561.1,578.1,588.6,623.4,617.1,626.5,
         628.1,640.7,645.6,654.1,658.1,676.4,681.4,689.4,699.5,700.8,708.2,725.7,
         729.1,738.4,739.6,740.1,737.3,736.1,735.2,713.7,725.1,723.3,715.2,699.3,
         698.6,700.7,689.3,667.6,655,668.9,647,638.3,628.6,609.8,611,595.2,574.9,
         543.9,515.7,472.8,436.9,417.2,398.1,372.8,360,327.4]

annee[11]=[415.5,457.1,493.8,523.4,554,557.6,600.5,621.2,645.1,679.7,630.4,620.7,
          707.8,734.3,730.7,751.5,770.8,783.1,790.5,795.3,800,805.9,807.4,810.4,
          812.2,814.3,811.2,808.8,727.3,762.4,758,765,747.8,745,776.8,778.9,
          766.7,777.7,754.9,750.6,750.2,705,675.4,687.9,684.9,662.7,652.6,634.2,
          613.7,587,557.5,513.2,466.2,419.8,380,315.9,252.9,155.7]

annee[12]=[695.5,729.5,770.1,801.3,830.6,850.4,867,887.8,902.7,914.1,920.5,925.9,
          941.9,945.9,954,961.8,968.7,975.2,979.6,978.9,983.1,982.4,984.7,987.8,
          993.8,991.2,989.2,983.8,989,990.5,982.4,982.7,977,978.1,976.4,977.5,
          965.1,963.7,954.4,956.3,955.2,927.3,909.8,907.8,897,883,868.5,848.6,
          832.9,816,791,759.4,723.4,686.6,642.5,583.9,514.7,362.7]


fen.mainloop()
```

ANNEXE 6 : DISPOSITIONS GENEREES PAR URAEUS POUR UNE HAUTEUR DE TOUR DE 9 METRES




Uraeus
SORTIES
Position :
Azimut : -77.16
Elévation : 87.58
Puissance au récepteur : 323.3 kW
Distance moyenne au récepteur : 20.48 m
Efficacité moyenne : 0.8
TYPE DE DISPOSITION
Circulaire quinquonce
Linéaire
Linéaire quinconce
ENTREES
Jour 21.
Mois 12.
Heure 12.
Minutes 0.
Secondes 0.
Hauteur de tour 9.
Inclinaison du récepteur
Dimension des héliostats 1.5
Coefficient de Reflexion 0.9
Nombre de facettes
Précision de visée
Valider
P(kW)
300
150
30
7H30
12H30
16H30
t(heures)
Janvier
Février
Mars
Avril
Mai
Juin
Juillet
Août
Septembre
Octobre
Novembre
Décembre
AFFICHER
Effet cosinus
Ombrages
Blocages
Ombrages + blocages
Efficacité globale
0.850
Début
Fin
0.7
Valider
0.9
Effacer graphe 1
Effacer graphe 2
AFFICHER
Disposition type
Disposition optimisée
Ressource
RECEPTEUR
Evolution puissance
Répartition puissance