



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
«Παραγωγή και Διαχείριση Ενέργειας»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Μοντελοποίηση οικιακής ζήτησης ηλεκτρικής ενέργειας για χρήση σε προσομοιώσεις: Μια υλοποίηση σε Python»

Του Μεταπτυχιακού Φοιτητή

Κοντοχριστόπουλον Ιωάννη

Επιβλέπων

**Δούκας Χρυσόστομος, Επίκουρος Καθηγητής, Σχολή Ηλεκτρολόγων
Μηχανικών και Μηχανικών Υπολογιστών**

I. Σύνοψη

Στον οικιακό τομέα καταναλώνεται ένα ιδιαίτερα σημαντικό ποσοστό της παγκόσμιας παραγωγής ενέργειας, περί το 20% του συνόλου αυτής, προκαλώντας αντίστοιχες εκπομπές CO₂. Για τη μείωση της κατανάλωσης ενέργειας, η πρόβλεψη της ζήτησης και κυρίως των αιχμών αυτής, έχει βασικότατο ρόλο. Προς αυτό το σκοπό, η καλύτερη ανάλυση της εξέλιξης της ζήτησης μπορεί να ανοίξει νέους δρόμους στο πεδίο της διεσπαρμένης παραγωγής, παρέχοντας δυνατότητες καλύτερου συγχρονισμού παραγωγής – ζήτησης και κάνοντας ρεαλιστική την εφαρμογή νέων τεχνολογιών όπως τα μικροδίκτυα. Σε αυτή τη μεταπτυχιακή εργασία παρουσιάζεται ένα μοντέλο προσομοίωσης της οικιακής ζήτησης ηλεκτρικής ενέργειας το οποίο υλοποιείται ως εφαρμογή ηλεκτρονικού υπολογιστή, γραμμένη στη γλώσσα προγραμματισμού Python. Το μοντέλο αποτελεί εξέλιξη προγενέστερων μοντέλων τα οποία βασίζονται στην ισχυρή σχέση που έχουν οι δραστηριότητες των κατοίκων με τη ζήτηση ηλεκτρικής ενέργειας σε κατοικίες. Πιο συγκεκριμένα, παράγονται δεδομένα ζήτησης ενέργειας με στοχαστικό τρόπο, κάνοντας χρήση τεχνικών Μαρκοβιανών αλυσίδων και μεθόδων Monte Carlo, λαμβάνοντας υπόψη τις δραστηριότητες των κατοίκων, τις εγκατεστημένες ηλεκτρικές συσκευές, τις ανάγκες φωτισμού καθώς και άλλους χρονικούς παράγοντες. Προστιθέμενη αξία στην εργασία αυτή δίνει η θεώρηση πως οι παράγοντες που επηρεάζουν τη συμπεριφορά των κατοίκων είναι ανεξάρτητοι μεταξύ τους αλλά και το γεγονός πως η υλοποίηση έγινε με τη μορφή εφαρμογής ηλεκτρονικού υπολογιστή, αυξάνοντας σημαντικά το πεδίο εφαρμογής της. Η εφαρμογή που αναπτύχθηκε έχει τη δυνατότητα δημιουργίας δεδομένων ζήτησης ισχύος ανάλυσης δεκαλέπτου, διατηρώντας ικανοποιητικές ταχύτητες εκτέλεσης και για μεγάλους αριθμούς κατοικιών. Συγκεκριμένα η μέθοδος είναι αποδοτική στη μοντελοποίηση συμπλεγμάτων κτιρίων μεσαίου μεγέθους και αναμένεται να έχει εφαρμογή στην προσομοίωση και τον προβλεπτικό έλεγχο μικροδίκτυων, αναλύοντας παράλληλα τις επιπτώσεις της εισαγωγής τους στην αγορά ηλεκτρικής ενέργειας. Στην εργασία παρουσιάζονται αποτελέσματα από την εκτέλεση της εφαρμογής τα οποία αποδεικνύουν την αξία της μεθόδου.

Λέξεις κλειδιά: οικιακή ζήτηση ηλεκτρικής ενέργειας, μοντελοποίηση ζήτησης; αλυσίδες μαρκόφ; διαχείριση της ζήτησης, μικροδίκτυα; python

II. Abstract

Residential housing is one of the most important energy consumption sectors, using about 20% of the global energy production, thus causing proportional CO₂ emissions. In order to reduce energy consumption, electricity demand prediction, specifically of the demand peaks is of great importance. Such prediction of the demand can open new paths concerning the application of innovative energy efficiency – low carbon strategies and techniques such as distributed production and demand side management (DSM) as well as the optimization of synchronization of production and demand, enabling the application of technologies such as microgrids. In this master's thesis, a model for domestic energy demand is presented and used as a basis for the development of a software application created with the Python programming language. The model is an evolution of pre-existing works and based on the strong correlation between domestic energy demand and the residents' activities. Specifically, per-house synthetic electricity demand data are created stochastically by using Markov Chain techniques and Monte Carlo methods, taking into account factors such as the residents' activities, the house appliances and luminaires and other temporal factors. The added value of this thesis, is in the assumption that the factors affecting the residents' behavior are independent, as well as the fact that a computer application was developed, improving its applicability. The developed application can generate ten-minute resolution power demand data, with fast execution times for even large numbers of residences. More specifically the application is efficient in modeling medium-sized residence clusters and expected to be of use for microgrid modeling and predictive control, also analyzing the impact of their introduction in the electrical energy market. In this study, generated results from the execution of the application are presented, confirming the value of the method.

Keywords: domestic electricity use; energy demand modelling; markov chains; demand side management; microgrids; python

III. Περίληψη

Η δυνατότητα πρόβλεψης της οικιακής ζήτησης ενέργειας αναμένεται στο μέλλον να είναι μεγάλης σημασίας στον τομέα της ενέργειας, με ιδιαίτερο όφελος για τις νέες τεχνολογίες και στρατηγικές ενεργειακής απόδοσης - χαμηλών εκπομπών ρύπων, όπως η διεσπαρμένη παραγωγή και η διαχείριση της ζήτησης (Demand Side Management - DSM). Τα περισσότερα σχετικά προβλεπτικά μοντέλα είναι μακροσκοπικά και βασίζονται συνήθως στη στατιστική ανάλυση της ζήτησης τόσο του πρόσφατου παρελθόντος (προηγούμενες ώρες/ημέρες/εβδομάδες), όσο και του μακρινότερου παρελθόντος (της αντίστοιχης χρονικής περιόδου σε προηγούμενα έτη), λαμβάνοντας παράλληλα υπόψη παράγοντες όπως τυχόν ευρύτερες τάσεις μεταβολής και εποχιακές διακυμάνσεις στη ζήτηση ηλεκτρικής ενέργειας. Μειονέκτημα αυτών των μεθόδων είναι αγνόηση μικροσκοπικών παραγόντων που επηρεάζουν τη βραχυπρόθεσμη ζήτηση ενέργειας, με πιθανή συνέπεια εσφαλμένες προβλέψεις όταν εξετάζονται δίκτυα μικρής κλίμακας ή απαιτούνται δεδομένα υψηλής χρονικής ανάλυσης. Επιπλέον, η επίλυση του προβλήματος με αυτόν τον τρόπο απαιτεί σημαντική επεξεργαστική ισχύ, η οποία δεν είναι αποδοτικό να χρησιμοποιείται για να λυθεί το ίδιο πρόβλημα κατ επανάληψη, πράγμα που συμβαίνει όταν εξετάζεται μεγάλος αριθμός ανεξάρτητων μικρών δικτύων. Σε αυτή τη μεταπτυχιακή εργασία παρουσιάζεται ένα μοντέλο προσομοίωσης της οικιακής ζήτησης ηλεκτρικής ενέργειας το οποίο υλοποιείται ως εφαρμογή ηλεκτρονικού υπολογιστή, γραμμένη στη γλώσσα προγραμματισμού Python και ικανή να παράγει δεδομένα ζήτησης ηλεκτρικής ενέργειας για κτίρια οικιακού τομέα. Το μοντέλο αποτελεί εξέλιξη προγενέστερων μοντέλων τα οποία βασίζονται στο γεγονός πως η ζήτηση ενέργειας μιας κατοικίας έχει στενή σχέση με τη χρήση των συσκευών της, η οποία μεταβάλλεται σημαντικά, εν γένει ανάλογα με τις δραστηριότητες των κατοίκων. Το πρόβλημα μοντελοποιείται προσομοιώνοντας τη συμπεριφορά και τις δραστηριότητες των κατοίκων, καθώς και τον τρόπο χρήσης των συσκευών και των φωτιστικών σωμάτων του κτιρίου.

Στη μοντελοποίηση χρησιμοποιήθηκαν τεχνικές Μαρκοβιανών Αλυσίδων, με τις οποίες περιγράφεται μια διεργασία βάσει των διαφορετικών διακριτών καταστάσεων στις οποίες βρίσκεται το εξεταζόμενο σύστημα. Διακρίνοντας μεταξύ των διαφορετικών καταστάσεων του συστήματος και με γνωστές τις πιθανότητες μετάβασης από τη μια κατάσταση στην άλλη, προσομοιώνεται η εξέλιξη της διεργασίας. Η μέθοδος αυτή αποτελεί υποκατηγορία της μεθόδου Μαρκοβιανών αλυσίδων με όνομα μέθοδος Μαρκοβιανών Αλυσίδων Μόντε Κάρλο (MCMC), διότι μοντελοποιεί το πρόβλημα ως μια Μαρκοβιανή αλυσίδα, στην οποία οι πιθανότητες μετάβασης έχουν υπολογιστεί με βάση στατιστικά δεδομένα. Η μέθοδος αυτή ενδείκνυται ιδιαίτερα για την ανάλυση του εξεταζόμενου προβλήματος, καθώς η κάθε κατάσταση στην οποία μπορεί να βρίσκεται μια κατοικία αποτελεί μια συγκεκριμένη κατάσταση ζήτησης ενέργειας, ενώ οι μεταβάσεις από τη μια κατάσταση στην άλλη αποτελούν 'γεγονότα' που οδηγούν σε μεταβολές στην κατανάλωση. Εν τέλει, η συνολική ζήτηση ενέργειας του κτιρίου εξαρτάται από τη ζήτηση ισχύος των συσκευών και φωτιστικών σωμάτων που περιλαμβάνει, η οποία προσομοιώνεται με τον τρόπο που αναφέρθηκε.

Όπως αναφέρθηκε, η μέθοδος στηρίζεται στη χρήση στατιστικών δεδομένων με τα οποία περιγράφεται η συμπεριφορά της ζήτησης ως προς τους παράγοντες που την επηρεάζουν. Στα δεδομένα αυτά απεικονίζονται παράγοντες ανθρώπινοι, όπως η συμπεριφορά, οι δραστηριότητες και οι συνήθειες των κατοίκων του κτιρίου, τεχνικοί, όπως ο τύπος και αριθμός των εγκατεστημένων συσκευών, αλλά και χρονικού/εποχιακοί, όπως η σχέση των δραστηριοτήτων των κατοίκων με την εποχή, τη μέρα, την ώρα και τον ηλιασμό. Στο μοντέλο έγινε η υπόθεση πως οι παράγοντες αυτοί είναι ανεξάρτητοι μεταξύ τους, ενώ από το συνδυασμό των προκύπτουν οι πιθανότητες μετάβασης ανάμεσα στις Μαρκοβιανές καταστάσεις. Η ακρίβεια των δεδομένων αυτών είναι πρωταρχικής σημασίας για την παραγωγή ορθών δεδομένων ζήτησης.

Στο μοντέλο, αρχικά υπολογίζεται η εξέλιξη του αριθμού των ενεργών κατοίκων (που αντιστοιχούν στους κατοίκους που μπορούν να εκτελέσουν μια δραστηριότητα την εξεταζόμενη στιγμή) μέσα στην ημέρα, και στη συνέχεια με βάσει αυτό τον αριθμό, προσομοιώνεται η χρήση των συσκευών του κτιρίου, η οποία οδηγεί σε συγκεκριμένη ζήτησης ηλεκτρικής ενέργειας για το κτίριο. Τόσο ο αριθμός των ενεργών κατοίκων όσο και η κατάσταση των συσκευών κάθε στιγμή, προσομοιώνονται ως Μαρκοβιανές αλυσίδες, και τα παραγόμενα δεδομένα ζήτησης ενέργειας έχουν ανάλυση δεκαλέπτου.

Ο αριθμός των ενεργών κατοίκων υπολογίζεται στοχαστικά για μια ζητούμενη χρονική περίοδο με χρήση μιας "μήτρας μεταβάσεων" στην οποία περιέχονται οι πιθανότητες μετάβασης από τη μια κατάσταση στην άλλη. Η μήτρα αυτή παράγεται με πολλαπλασιασμό πινάκων, καθένας από τους οποίους αντιπροσωπεύει έναν από τους παράγοντες που επηρεάζουν τον αριθμό των ενεργών κατοίκων. Τέτοιοι παράγοντες είναι η ώρα της

ημέρας, ο τύπος της ημέρας (καθημερινή ή αργία), ο αριθμός των ενεργών κατοίκων στην προηγούμενη κατάσταση και ο συνολικός αριθμός κατοίκων του κτιρίου. Κάνοντας χρήση αυτού του πίνακα μεταβάσεων σε συνδυασμό με μια γεννήτρια ψευδοτυχαίων αριθμών υπολογίζονται οι μεταβάσεις από κατάσταση σε κατάσταση.

Στη συνέχεια ο αριθμός των ενεργών κατοίκων χρησιμοποιείται ως δεδομένο τον υπολογισμό του τρόπου και της συχνότητας χρήσης των συσκευών του κτιρίου. Κάθε συσκευή έχει χαρακτηριστικά που την προσδιορίζουν όπως για παράδειγμα η συχνότητα χρήσης της ανάλογα με την ώρα της ημέρας, η ζήτηση ισχύος σε κατάσταση λειτουργίας και αναμονής, οι τυπικοί χρόνοι λειτουργίας και αναμονής κ.α. Η κατάσταση λειτουργίας των συσκευών του κτιρίου αναπαρίσταται ως Μαρκοβιανή αλυσίδα, της οποίας η κάθε κατάσταση αντιστοιχεί σε συγκεκριμένο συνδυασμό καταστάσεων των συσκευών και συνεπώς σε συγκεκριμένη ζήτηση ενέργειας. Με αυτή τη μέθοδο, αρκεί σε κάθε χρονική στιγμή να είναι γνωστή η κατάσταση λειτουργίας κάθε συσκευής καθώς και οι παράγοντες που οδηγούν σε έναυστη ή σβέση αυτών για να υπολογιστεί η συνολική ζήτηση ενέργειας τους. Τα φωτιστικά σώματα ανήκουν στην ευρύτερη κατηγορία των συσκευών, εξετάζονται όμως ξεχωριστά, διότι ο τρόπος λειτουργίας τους διαφέρει σημαντικά από αυτόν των άλλων συσκευών. Η χρήση τους εξαρτάται τόσο από τη στάθμη του εξωτερικού φωτισμού όσο και από τη δραστηριότητα των κατοίκων, ενώ ο κύκλος λειτουργίας τους διαφέρει αυτόν των άλλων συσκευών διότι σχετίζεται απευθείας με την ύπαρξη ή μη ενεργών κατοίκων στο δωμάτιο. Η λειτουργία των φωτιστικών μοντελοποιείται επίσης ως Μαρκοβιανή αλυσίδα, με κάθε συνδυασμό καταστάσεων λειτουργίας όλων των φωτιστικών σωμάτων (ενεργά/ανενεργά) να αποτελεί μια Μαρκοβιανή κατάσταση. Η μετάβαση από τη μια κατάσταση στην άλλη εξαρτάται από τον εξωτερικό φωτισμό, τον αριθμό των ενεργών κατοίκων και ένα συντελεστή ‘χρήσης’ του φωτιστικού που απεικονίζει τη διαφορετική πιθανότητα χρήσης που έχουν τα φωτιστικά ανάλογα με το σημείο που έχουν εγκατασταθεί. Στο μοντέλο λαμβάνεται επίσης υπόψη το φαινόμενο της ‘κοινής χρήσης φωτιστικών’, δηλαδή το φαινόμενο της τάσης προς συνύπαρξη των κατοίκων του κτιρίου στο φωτισμένο δωμάτιο, εισάγοντας την έννοια των ‘πραγματικών ενεργών κατοίκων’, δηλαδή των ενεργών κατοίκων που έχουν ανάγκη για φωτισμό από διαφορετικά φωτιστικά σώματα ο καθένας. Εν τέλει, με χρήση όσων που προαναφέρθηκαν, παράγονται συνθετικά δεδομένα ζήτησης ηλεκτρικής ισχύος, μετά από αναλυτική μοντελοποίηση της συμπεριφοράς κατοίκων, συσκευών και φωτιστικών.

Βάσει των ανωτέρω, παράγεται η καμπύλη ζήτησης ηλεκτρικής ισχύος από ένα κτίριο στο διάστημα που εξετάζεται. Η ζήτηση μόνο ενός προσομοιωμένου κτιρίου δεν αρκεί για την εξαγωγή ασφαλών συμπερασμάτων σχετικά με το εξεταζόμενο δίκτυο, αλλά με χρήση του μοντέλου για προσομοίωση της ζήτησης πολλών ανεξαρτήτων κτιρίων, μπορούν να εξαχθούν ασφαλή δεδομένα σε σχέση με τη συνολική ζήτηση ενός συμπλέγματος κτιρίων.

Η υλοποίηση του μοντέλου ως εφαρμογή έγινε με χρήση της γλώσσας προγραμματισμού Python και χρησιμοποιώντας τεχνικές σπονδυλωτού (modular) και αντικειμενοστραφή (object-oriented) προγραμματισμού, τηρώντας ωστόσο τεχνικές ορθού προγραμματισμού και επαναχρησιμοποίησης κώδικα και πετυχαίνοντας ικανοποιητικές ταχύτητες εκτέλεσης. Τα δεδομένα που εισάγονται στην εφαρμογή ή εξάγονται από αυτήν είναι τύπου .csv ή .xlsx, λόγω της καλής συμβατότητας των μορφών αυτών με άλλες εφαρμογές και της ευκολίας στην οπτικοποίηση δεδομένων αλλά και περεταίρω επεξεργασία που αυτές παρέχουν. Στην εφαρμογή έγινε εκτεταμένη χρήση των βιβλιοθηκών της python “numpy” και “pandas”.

Βρέθηκε πως τα παραγόμενα δεδομένα συσχετίζονται πολύ ευνοϊκά με πραγματικά δεδομένα, συνεπώς το μοντέλο είναι ικανό να προσομοιώσει τη ζήτηση ενέργειας από κτίρια οικιακού τομέα, ο οποίος ήταν και ο αρχικός στόχος. Η εφαρμογή που αναπτύχθηκε έχει τη δυνατότητα δημιουργίας δεδομένων ζήτησης ισχύος για μεγάλους αριθμούς κατοικιών διατηρώντας ικανοποιητικές ταχύτητες εκτέλεσης. Η θεώρηση πως οι παράγοντες που επηρεάζουν τη συμπεριφορά των κατοίκων είναι ανεξάρτητοι μεταξύ τους αποτελεί δυνατό σημείο της εργασίας, καθώς δίνει τη δυνατότητα μοντελοποίησης αλλαγών σε έναν από τους παράγοντες αυτούς, οι οποίες θα μπορούσαν να προκύψουν από συμπεριφορικές αλλαγές. Επιπλέον, το γεγονός πως η εφαρμογή είναι γραμμένη σε μια δημοφιλή, εύκολα επεκτάσιμη και εύκολη στη χρήση γλώσσα προγραμματισμού όπως η python, της δίνει ιδιαίτερη αξία, καθώς είναι εύκολη η ενσωμάτωσή της σε εφαρμογές μεγαλύτερης κλίμακας που απαιτούν δεδομένα ζήτησης ενέργειας υψηλής ανάλυσης, όπως για παράδειγμα προσομοιώσεις και αναλύσεις εφαρμογών διαχείρισης ζήτησης (DSM) ή μικροδικτύων.

Λέξεις κλειδιά: οικιακή ζήτηση ηλεκτρικής ενέργειας, μοντελοποίηση ζήτησης; αλυσίδες Μαρκόβ; διαχείριση της ζήτησης, μικροδικτυα; python

IV. Summary

Being able to forecast residential power demand is expected to be in the future of significant importance to the field of energy production, with new technologies and energy efficiency-low carbon strategies such as distributed generation and Demand Side Management (DSM) particularly benefitting from such data. Most power demand forecasting models are macroscopic and are usually based on the statistical analysis of demand in the short-term (within latest days/weeks) and long-term (during the same time period over a range of earlier years), also taking into account factors such as trends and seasonalities affecting demand. A possible weakness of such methods is their failure to take into account factors microscopic in nature that can affect the short-term power demand, generating possibly mistaken forecasts when considering small scale grids or when high resolution data is required. Besides, solving the problem in this way can require considerable processing power, thus using such methods to generate power demand data for multiple independent small scale grids can be inefficient. In this master's thesis, a model for domestic energy demand is presented and used as a basis for the development of a software application created with the Python programming language. The model is an evolution of pre-existing works and based upon the fact that residential power demand is closely related to the use of a residence's appliances, which in turn is in close relation to the activities of the residents. The problem is modeled by simulating the behavior and activities of the residents, as well as the consequent use of the appliances and luminaires of the residence.

Markov chain techniques are of particular value to such a model, since such techniques can describe a process based on the specific states where the considered system can be at any point in time. By knowing the probability of a state changing into a specific other state, the process can be represented. This particular methodology is a subcategory of the Markov Chains method, named Markov Chain Monte Carlo (MCMC), since it models the process as a Markov Chain, to which the probabilities of moving to a new state have been calculated by using statistical data. This method is particularly suitable for analyzing the problem at hand since each state can be mapped to a specific power demand state of the residence while the transition for a state to another is an event, for which the probability to happen is known, depending on the several factors considered.

In the model we define as 'residents' the total number of a house's occupants, while 'active residents' at each time period are only those occupants that can initiate an activity within that specific time period. The behavior and activities of the active residents constitute one of the most important factor influencing residential power demand and they in close relation to the use of the residential appliances. We define as 'appliance' any electrical appliance that can contribute to the power demand of a residence, while luminaires are a subcategory of appliances. In the end, the total power demand of a residence is strongly related to the power demand of the appliances it contains.

The implementation is based on the use of statistical data to describe the power demand behavior with regards to the factors that influence it. Multiple factors are represented within this data, human-based, such as the behavior and activities of the residents, technical-based, such as the number and type of installed appliances or temporal-based, such as the correlation of the resident's activities with the season, specific day and hour, or irradiance. A model assumption is that those factors are independent from each other, and that their combination leads to the Markov states' transition probabilities. The accuracy of such data is of primary importance for generating accurate power demand data.

Within the model, the number of active residents (residents that can have an activity at the studied point in time) is initially calculated for a whole day, and then by using this value the use of the houses' appliances is modeled, leading to a specific electricity demand for the building. The number of active residents and the appliances states are modeled as Markov chains, while the generated data have a 10 minute resolution.

The number of active residents is generated stochastically for each required time period by using a "transitional matrix" containing the probability of the current number of active residents changing to a new state. This matrix is produced by matrix multiplication, each matrix representing a factor influencing the number of active residents. Such factors are the type of the day (workday or weekend), the time of day, the last state (number of active residents in the last period) and the total number of occupants. By using this transitional matrix in conjunction with a pseudorandom number generator, transitions from each state to the next are determined.

Subsequently, the number of active residents is used as input data in order to calculate the manner and frequency of use of the house appliances. Each appliance is characterized by a number of attributes such as the

frequency of use depending on the time of the day, the “on” and “standby” power demand, the typical duration of “on” and “off” states, etc. The appliances’ status is represented as a Markov chain with each state corresponding to a specific combination of each appliance work-status and thus to a particular total power demand. In this way, by knowing the state of each appliance at every moment, as well as the factors that lead to an appliance powering on or off suffices to calculate the total power demand for the residence. The luminaires of the residence are not included in this calculation and modeled differently since, even though they are a type of appliance, their power-on and power-off behavior is considerably different to that of other appliances. Specifically, their use is dependent not only to the activities of the occupants but also to the external lighting levels, while their power-on cycles are directly related to the existence of active residents within the same space. The state of the luminaires is also modeled as a Markov chain, with each combination of the luminaires work-states representing a specific overall state. Transitions from a state to another are influenced by the external lighting levels, the number of active residents and a ‘use coefficient’ assigned independently to each luminaire, describing the difference in use of a luminaire depending on its position within the residence. Also taken into account in the model is the tendency for light sharing from residents in the same space, made possible by introducing the concept of “effective active residents”, describing the active residents who require an independent light source. Finally, by integrating together all the concepts mentioned above, synthetic power demand data is generated, after extensive simulation of the behavior and state of the occupants, appliances and luminaires of a residence.

Based on the above, it is possible to generate a power demand curve for a residence within a requested time period. This curve on its own is not enough to lead to accurate conclusions, but by using the model for simulating the power demand of multiple independent buildings we can generate reliable and statistically accurate demand data for a building cluster.

The model was implemented as a software application written in the programming language Python by using modular and object-oriented (OO) programming techniques, aiming for code efficiency, reusability and readability, achieving fast execution times and an adaptable and robust codebase. The input and output data can be of .csv or .xlsx format, these chosen due to their compatibility with existing popular software applications and their ease of visualization. In the application, the python libraries ‘numpy’ and ‘pandas’ were used extensively, providing performance optimizations, advanced data structures and data analysis tools.

The generated data was found to be similar to real-world data, thus proving the model’s ability to simulate the power demand process of residences and achieving the target of the study. The developed software can generate power demand data for large number of residences with reasonably fast execution times. The assumption that the factors affecting resident behavior are independent is a strong point of the study, as can enable modeling the change of one of those factors, resulting from a behavioral change. In addition, the application can be used to model building clusters, provided detailed statistical data for the area is available, thus transforming the problem of power demand forecast to the problem of identifying the resident behavior and installed applications technology in an area, which can be simple to acquire via surveys. Finally, the fact that the software application is written in a popular, extensible and reusable programming language such as python, could be of significant value in the use of this model within broader-scale applications requiring high-resolution residential demand data, such as DSM and microgrid simulations and analyses

Keywords: domestic electricity use; energy demand modelling; markov chains; demand side management; microgrids; python

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον καθηγητή μου, κ. Χάρη Δούκα, επιβλέποντα της παρούσας μεταπτυχιακής εργασίας, για την ευκαιρία που μου παρείχε για την εκπόνηση μιας ιδιαίτερα ενδιαφέρουσας μελέτης στο πεδίο της πρόβλεψης της ζήτησης και της βελτίωσης της ενεργειακής αποδοτικότητας, ενός τομέα ιδιαίτερα σημαντικού για το μέλλον. Η καθοδήγησή του και η άριστη συνεργασία που είχαμε ήταν βασικής σημασίας για την εκπόνηση της εργασίας αυτής.

Επίσης, θα ήθελα να ευχαριστήσω τον κ. Αλέξανδρο Φλάμο, καθηγητή στο Πανεπιστήμιο Πειραιώς, καθώς και τον κ. Σωτήρη Παπαδέλη, ερευνητή στο Πανεπιστήμιο Πειραιώς, από τους οποίους επίσης είχα καθοδήγηση και των οποίων η συμβολή στην εργασία αυτή ήταν καθοριστική. Η υποστήριξή τους καθ' όλη τη διάρκεια της εργασίας ήταν ουσιαστική και πολύπλευρη, ενώ οι συμβουλές τους με έκαναν σε πολλά σημεία να αναθεωρήσω την προσέγγισή μου ερευνώντας το ζήτημα εις βάθος, προς όφελος του τελικού αποτελέσματος.

Περιεχόμενα

1.	Εισαγωγή.....	10
1.1.	Ανάλυση προβλήματος.....	10
1.2.	Προγενέστερη έρευνα.....	11
1.3.	Μαρκοβιανές αλυσίδες.....	11
1.4.	Σχεδίαση από κάτω προς τα πάνω (bottom up)	12
1.5.	Στατιστικά δεδομένα	13
2.	Μοντέλο	14
2.1.	Φιλοσοφία προσέγγισης.....	14
2.2.	Ενεργοί Κάτοικοι	14
2.3.	Συσκευές.....	18
2.4.	Φωτισμός	21
3.	Υλοποίηση	25
3.1.	Γενικά	25
3.2.	Κύρια Κλάση (main)	26
3.3.	Κτίριο (house).....	27
3.4.	Δεδομένα Συσκευών (appliance_data)	27
3.5.	Δεδομένα Φωτιστικών (luminaire_data)	28
3.6.	Δεδομένα Ενεργών Κατοίκων (occupancy_data)	28
3.7.	Ενεργοί Κάτοικοι (occupancy)	29
3.8.	Συσκευές (appliance)	29
3.9.	Ζήτηση ισχύος συσκευών (electricity).....	30
3.10.	Φωτιστικό Σώμα (luminaire).....	30
3.11.	Φωτισμός (lighting).....	31
3.12.	Κοινή βιβλιοθήκη (shared)	31
3.13.	Απόδοση εφαρμογής.....	31
4.	Αποτελέσματα - Συζήτηση	33
4.1.	Παρουσίαση αποτελεσμάτων	33
4.2.	Συζήτηση.....	41
5.	Συμπεράσματα	43
6.	Επόμενα βήματα	45
7.	Βιβλιογραφία.....	47
8.	Παράτημα.....	49
8.1.	Κώδικας προγράμματος	49
8.1.1.	Main	49
8.1.2.	House	51
8.1.3.	Occupancy_data	52
8.1.4.	Appliance_data	54
8.1.5.	Luminaire_data	58
8.1.6.	Appliance.....	58
8.1.7.	Luminaire	61
8.1.8.	Electricity	63
8.1.9.	Lighting	64
8.1.10.	Shared	65

1. Εισαγωγή

Η πρόβλεψη της ζήτησης ηλεκτρικής ενέργειας αποτελεί ένα ιδιαίτερα σημαντικό ζήτημα. Χωρίς αυτή είναι αδύνατη η μετέπειτα αποδοτική διαχείριση της παραγωγής ηλεκτρικής ενέργειας, με επακόλουθο τη σπατάλη ενέργειας. Σε αντίθεση, η ακριβής πρόβλεψη της ζήτησης έχει πολλαπλά οφέλη. Διευκολύνει το συγχρονισμό παραγωγής και ζήτησης με τον παραδοσιακό τρόπο και δίνει τη δυνατότητα εφαρμογής μιας σειράς από καινοτόμες τεχνικές εξοικονόμησης ενέργειας και μείωσης εκπομπών διοξειδίου του άνθρακα. Παραδείγματα τέτοιων τεχνικών είναι η διαχείριση της ζήτησης (Demand Side Management) [1] [2] και τα μικροδίκτυα. [3] [4] Παράλληλα ευνοείται η περεταίρω αύξηση της εκμετάλλευσης ανανεώσιμων πηγών ενέργειας, διότι περιορίζεται το πρόβλημα που προκαλείται λόγω της στοχαστικότητας της παραγωγής ενέργειας από τέτοιες πηγές.

Σε αυτή τη μεταπτυχιακή εργασία παρουσιάζεται και υλοποιείται με τη μορφή εφαρμογής ηλεκτρονικού υπολογιστή ένα μοντέλο πρόβλεψης της οικιακής ζήτησης ηλεκτρικής ενέργειας. Το μοντέλο βασίζεται στην προσομοίωση των δραστηριοτήτων των κατοίκων κάθε κατοικίας οι οποίες οδηγούν σε χρήση συσκευών που καταναλώνουν ενέργεια. Στην προσομοίωση λαμβάνονται υπόψη παράγοντες ανθρώπινοι, όπως η συμπεριφορά, οι δραστηριότητες και οι συνήθειες των κατοίκων του κτιρίου, τεχνικοί, όπως ο τύπος και αριθμός των εγκατεστημένων συσκευών, αλλά και χρονικοί/εποχιακοί, όπως η σχέση των δραστηριοτήτων των κατοίκων με την εποχή, τη μέρα, την ώρα και τον ηλιασμό του κτιρίου. Το μοντέλο αποτελεί εξέλιξη των μοντέλων που παρουσιάστηκαν από τους Richardson, I. et al. [5] [6] [7], στα οποία έχουν γίνει διάφορες βελτιώσεις και βελτιστοποιήσεις με στόχο την ακριβέστερη και αποδοτικότερη λειτουργία.

Προστιθέμενη αξία σε αυτή την εργασία δίνει η θεώρηση πως οι παράγοντες που επηρεάζουν τη συμπεριφορά των κατοίκων είναι ανεξάρτητοι μεταξύ τους. Η σύνθεση αυτών αποτελεί τη βάση για τη μοντελοποίηση της συμπεριφοράς τους, ενώ πιθανές συμπεριφορικές αλλαγές μπορούν να μοντελοποιηθούν με μεταβολές σε ορισμένους από τους παράγοντες αυτούς. Επιπλέον, ιδιαίτερα σημαντικό είναι το γεγονός πως η υλοποίηση έγινε με τη μορφή εφαρμογής ηλεκτρονικού υπολογιστή, η οποία αυξάνει σημαντικά το πεδίο εφαρμογής του μοντέλου.

1.1. Ανάλυση προβλήματος

Το υπό μελέτη πρόβλημα είναι η πρόβλεψη της ζήτησης ηλεκτρικής ενέργειας από μια κατοικία για ένα ορισμένο χρονικό διάστημα. Συγκεκριμένα είναι βασικής σημασίας η πρόβλεψη της εξέλιξης της ζήτησης ισχύος στο διάστημα που εξετάζεται. Η πρόβλεψη αυτή δεν είναι εύκολο να επιτευχθεί, καθώς η ζήτηση μιας κατοικίας επηρεάζεται από ένα μεγάλο αριθμό παραγόντων όπως για παράδειγμα την ώρα της ημέρας, την εποχή του έτους, τον αριθμό των κατοίκων, τις συνήθειες και δραστηριότητές τους, την περιοχή την οποία μελετάμε κ.α. Επιπλέον, το είδος και ο αριθμός των εγκατεστημένων συσκευών και φωτιστικών της κατοικίας, είναι βασικής σημασίας για τη ζήτηση ενός κτιρίου. Τα περισσότερα προβλεπτικά μοντέλα είναι μακροσκοπικά μεγάλης κλίμακας (πχ. [8] [9]) και βασίζονται συνήθως στη στατιστική ανάλυση της ζήτησης ενέργειας τόσο στο πρόσφατο παρελθόν (προηγούμενες ώρες/ημέρες/εβδομάδες), όσο και στο μακρινότερο παρελθόν (την

αντίστοιχη χρονική περίοδο στα προηγούμενα έτη). Σε τέτοια μοντέλα λαμβάνονται επίσης υπόψη παράγοντες όπως τυχόν ευρύτερες τάσεις αύξησης/μείωσης στη ζήτηση και εποχιακές διακυμάνσεις στο κόστος της ενέργειας. Μειονέκτημα αυτών των μεθόδων είναι η μη θεώρηση των μικροσκοπικών παραγόντων που αναφέρθηκαν παραπάνω, με πιθανή συνέπεια εσφαλμένες προβλέψεις όταν εξετάζονται δίκτυα μικρής κλίμακας. Επιπρόσθετα, η επίλυση του προβλήματος με αυτόν τον τρόπο απαιτεί σημαντική επεξεργαστική ισχύ, η οποία δεν είναι αποδοτικό να χρησιμοποιείται για να λυθεί κατ' επανάληψη το ίδιο πρόβλημα για μεγάλους αριθμούς ανεξαρτήτων δικτύων μικρής κλίμακας. Αντίθετα με αυτά τα μοντέλα, η παρούσα εργασία επικεντρώνεται στη σύνδεση της οικιακής ζήτησης ενέργειας με τη συμπεριφορά, τις δραστηριότητες, τις συνήθειες και τον τρόπο ζωής των κατοίκων.

1.2. Προγενέστερη έρευνα

Έρευνα στον τομέα της μοντελοποίησης οικιακής ζήτησης βασισμένη στους κατοίκους των κτιρίων που εξετάζονται υπάρχει ήδη. Στην εργασία των Capasso et al. [10] εμφανίζεται αρχικά η ιδέα της μοντελοποίησης οικιακής ζήτησης βάσει της συμπεριφοράς κατοίκων και των εγκατεστημένων συσκευών του κτιρίου και αποδεικνύεται η αξία της μεθόδου αυτής. Διάφορα άλλα αντίστοιχα μοντέλα, τα οποία έχουν παραλληλισμούς με αυτά που χρησιμοποιούνται στην εργασία αυτή έχουν παρουσιαστεί, όπως αυτά από τους Paatero και Lund [11], Stokes et al. [12], Widén and Wäckelgård [13], Wilke et al. [14], Yao and Steemers [15]. Επίσης, όπως προαναφέρθηκε, η εργασία αυτή έχει βασιστεί στα μοντέλα πρόβλεψης χρήσης κτιρίου (occupancy), συσκευών, και φωτισμού που παρουσιάστηκαν από τους Richardson I., et al. [5] [6] [7], στα οποία γίνεται προσομοίωση της οικιακής ζήτησης βάσει μιας σειράς σχετικών μικροσκοπικών παραγόντων.

1.3. Μαρκοβιανές αλυσίδες

Για τη μοντελοποίηση χρησιμοποιούνται σε μεγάλο βαθμό τεχνικές Μαρκοβιανών Αλυσίδων διακριτού χρόνου [16] [17], με τις οποίες περιγράφεται μια στοχαστική διεργασία με βάση τη σειριακή ακολουθία καταστάσεων οι οποίες περιγράφουν το σύστημα στο οποίο εκτελείται η διεργασία αυτή κάθε χρονική στιγμή. Αρχικά ορίζονται οι καταστάσεις στις οποίες είναι δυνατόν να βρεθεί το σύστημα και στη συνέχεια η διεργασία μοντελοποιείται ως μια ακολουθία καταστάσεων με μεταβάσεις από κατάσταση σε κατάσταση (όπου η επόμενη κατάσταση μπορεί να συμπίπτει με την προηγούμενη). Με αυτόν τον τρόπο, καθώς έχουμε μεταβάσεις σε κάθε διακριτή χρονική στιγμή, προσομοιώνεται στοχαστικά η εξέλιξη της διεργασίας στο χρόνο. Στην εργασία αυτή, οι βασικές πιθανότητες μετάβασης έχουν βρεθεί ως αποτέλεσμα στατιστικής έρευνας πάνω στις διεργασίες που προσομοιώνονται, και σχετίζονται κυρίως με τη συμπεριφορά των κατοίκων και άλλους παράγοντες που επηρεάζουν τη χρήση των συσκευών του κτιρίου.

Η μέθοδος αυτή αποτελεί υποκατηγορία της μεθόδου Μαρκοβιανών αλυσίδων με όνομα μέθοδος Μαρκοβιανών Αλυσίδων Monte Carlo (MCMC) [16] [18], διότι μοντελοποιεί το πρόβλημα ως μια Μαρκοβιανή αλυσίδα, στην οποία οι πιθανότητες μετάβασης έχουν

υπολογιστεί με βάση στατιστικά δεδομένα. Είναι ιδανική για την ανάλυση του εξεταζόμενου προβλήματος, καθώς κάθε κατάσταση στην οποία βρίσκεται η κατοικία αποτελεί μια διακριτή κατάσταση ζήτησης ενέργειας, ενώ οι μεταβάσεις από τη μια κατάσταση στην άλλη μπορούν να θεωρηθούν “γεγονότα” τα οποία προκαλούνται από τους παράγοντες που επηρεάζουν τη ζήτηση και των οποίων η πιθανότητα να συμβούν μπορεί να βρεθεί με στατιστική ανάλυση.

Στο μοντέλο που χρησιμοποιείται δε γίνεται απευθείας υπολογισμός αυτής της αλυσίδας, διότι είναι γνωστές οι πιθανότητες μετάβασης σε κάθε διακριτή κατάσταση ζήτησης ενέργειας του κτιρίου. Αντίθετα, η κατάσταση όλων των συσκευών και των φωτιστικών ορίζει μια ζήτηση ενέργειας κάθε στιγμή. Η εξέλιξη καθεμιάς από αυτές τις καταστάσεις είναι επίσης μια Μαρκοβιανή αλυσίδα, στην οποία οι πιθανότητες μετάβασης στην επόμενη κατάσταση υπολογίζονται σε κάθε χρονικό σημείο ως αποτέλεσμα της προσομοίωσης. Οι πιθανότητες αυτές, επηρεάζονται από την εξέλιξη των παραγόντων που οδηγούν στα γεγονότα έναυσης/σβέσης των συσκευών, όπως πχ. η παρουσία ή μη των κατοίκων στο σπίτι. Η εξέλιξη αυτών των βασικών παραγόντων μοντελοποιείται επίσης με τη μέθοδο των Μαρκοβιανών αλυσίδων, με τη διαφορά ότι οι πιθανότητες μετάβασης αυτών των αλυσίδων είναι γνωστές. Ως αποτέλεσμα αυτής της σχεδίασης, το μοντέλο αποτελείται από αλυσίδες των οποίων η κατάσταση “οδηγεί” άλλες αλυσίδες. Για το λόγο αυτό, η προσομοίωση απαιτείται να γίνει σειριακά, καθώς η κάθε κατάσταση μιας αλυσίδας ανώτερου επιπέδου εξαρτάται από την προηγούμενη κατάστασή της, αλλά και από την κατάσταση των άλλων αλυσίδων (κατώτερου επιπέδου) σε αυτό το χρονικό σημείο, οι οποίες πρέπει να βρεθούν προηγουμένως.

1.4. Σχεδίαση από κάτω προς τα πάνω (bottom up)

Η μέθοδος που ακολουθήθηκε για την υλοποίηση περιγράφεται καλύτερα ως σχεδίαση “bottom up”, δηλαδή από κάτω προς τα πάνω [19]. Το συνολικό πρόβλημα χωρίζεται σε υποτμήματα και λύνεται ως σύνθεση των λύσεων που παράγονται από τη λύση των υποτμημάτων. Τα υποτμήματα λειτουργούν ανεξάρτητα μεταξύ τους, είτε παράλληλα είτε σειριακά (το ένα μπορεί να δίνει ή να μη δίνει δεδομένα στο άλλο). Πλεονέκτημα αυτής της μεθόδου είναι πως προωθείται η ανάπτυξη μικρών τμημάτων του μοντέλου (και της εφαρμογής) κάθε φορά, τα οποία μπορούν να αναπτυχθούν και να αξιολογηθούν γρήγορα εφόσον είναι ανεξάρτητα μεταξύ τους και περιορισμένα σε έκταση. Παράλληλα είναι εύκολη η αντιμετώπιση πιθανών νέων αναγκών για τη λύση του ευρύτερου προβλήματος, απλά με προσθήκη νέων υποτμημάτων. Έτσι επιτυγχάνεται ανάπτυξη σε στέρεα βάση και μεγάλη προσαρμοστικότητα κατά την ανάπτυξη, καθώς βελτιώσεις και προσθήκες στο μοντέλο μπορούν να επιτευχθούν χωρίς την ανάγκη ευρύτερου επανασχεδιασμού. Επιπλέον, λόγω της επιλογής αυτής, προγραμματιστικά προωθείται η σπονδυλωτή λογική και ο αντικειμενοστραφής προγραμματισμός, που είναι εγνωσμένης αξίας όσον αφορά στη γρήγορη ανάπτυξη, προσαρμοστικότητα και δυνατότητα επαναχρησιμοποίησης μιας εφαρμογής. Τα εξ’ ορισμού μειονεκτήματα αυτής της τακτικής είναι η πιθανή αυξημένη ζήτηση πόρων για την προσομοίωση, καθώς και ο αυξημένος χρόνος σχεδίασης και ανάπτυξης. Η πρώτη αδυναμία παρακάμφηκε με την προσεκτική σχεδίαση και χρήση

αποδοτικών αλγορίθμων στο μοντέλο, καθώς και με αποδοτικό προγραμματισμό, ενώ για τη δεύτερη ο χαμένος χρόνος ανακτήθηκε από τα οφέλη στην αξιολόγηση και εκσφαλμάτωση (debugging) του κώδικα λόγω της περιορισμένης έκτασης των υποτυπωμάτων που αναπτύχθηκαν ανεξάρτητα μεταξύ τους.

1.5. Στατιστικά δεδομένα

Εφόσον η μοντελοποίηση του προβλήματος γίνεται με βάση τους παράγοντες που το επηρεάζουν (οι οποίοι έχουν σχέση με γεγονότα που οδηγούν σε ένανση/σβέση των συσκευών και φωτιστικών του κτιρίου), η ακρίβεια με την οποία μπορούν να αναγνωριστούν και να αναλυθούν οι παράγοντες αυτοί είναι βασικής σημασίας για την ακρίβεια των αποτελεσμάτων. Στο συγκεκριμένο μοντέλο χρησιμοποιούνται στατιστικά δεδομένα για την εξέταση της συμπεριφοράς της ζήτησης ως προς τους παράγοντες αυτούς. Τα δεδομένα αυτά εκφράζουν ανθρώπινους παράγοντες όπως η συμπεριφορά, οι δραστηριότητες και οι συνήθειες των κατοίκων του κτιρίου, τεχνικούς παράγοντες όπως το εγκατεστημένο υλικό στο κτίριο και χρονικούς παράγοντες όπως η σχέση των δραστηριοτήτων με την εποχή, την ώρα, και το φως του ήλιου. Η ακρίβειά τους είναι πρωταρχικής σημασίας για την παραγωγή δεδομένων καλής ακρίβειας (τα οποία συγκλίνουν με πραγματικά), αλλά επίσης δίνουν σημαντική ευελιξία στο μοντέλο. Αυτό γίνεται διότι με πιθανή χρήση στατιστικών δεδομένων που εκφράζουν διαφορετικές περιοχές, καταστάσεις, καιρικά φαινόμενα ή συμπεριφορικές αλλαγές, είναι δυνατό να γίνουν προβλέψεις οι οποίες θα ανταποκρίνονται στο εκάστοτε συγκεκριμένο πρόβλημα. Έτσι, αλλάζοντας μόνο τα στατιστικά δεδομένα εισόδου, το μοντέλο μπορεί να εφαρμοστεί σε ένα ευρύ πεδίο διαφορετικών εκδοχών του προβλήματος ή συνδυασμούς αυτών. Στην παρούσα υλοποίηση, λόγω έλλειψης δεδομένων για τον ελληνικό χώρο, χρησιμοποιήθηκαν αντίστοιχα δεδομένα Μ. Βρετανίας, ενώ έγιναν εκτιμήσεις για ορισμένα δεδομένα τα οποία ήταν αδύνατο να αποκτηθούν.

2. Μοντέλο

2.1. Φιλοσοφία προσέγγισης

Όπως προαναφέρθηκε, η διεργασία που μοντελοποιείται είναι η εξέλιξη της ζήτησης ηλεκτρικής ισχύος σε μια κατοικία. Για να υπολογιστεί, αρχικά υπολογίζεται η εξέλιξη του αριθμού των “ενεργών κατοίκων” (οι κάτοικοι που μπορούν να έχουν κάποια δραστηριότητα) σε κάθε διακριτή χρονική στιγμή. Ο αριθμός αυτός εξαρτάται από διάφορους παράγοντες οι οποίοι θα αναλυθούν στη συνέχεια. Επίσης, ο αριθμός αυτός αποτελεί έναν από τους παράγοντες που επηρεάζουν τη χρήση των συσκευών και φωτιστικών της κατοικίας. Στη συνέχεια, λαμβάνοντας υπόψη τον αριθμό των “ενεργών κατοίκων” αλλά και άλλους παράγοντες (όπως τις ανάγκες των κατοίκων ανάλογα με τη χρονική στιγμή, την κατάσταση του εξωτερικού φωτισμού κ.α.), γίνεται προσομοίωση της χρήσης των συσκευών και φωτιστικών της κατοικίας από τους κατοίκους. Από τη χρήση αυτή υπολογίζεται η ζήτηση ισχύος κάθε συσκευής αλλά και η συνολική ζήτηση ισχύος της κατοικίας σε κάθε διακριτή χρονική στιγμή εντός του εξεταζόμενου χρονικού διαστήματος.

Το μοντέλο λειτουργεί με ανάλυση δεκαλέπτου, συνεπώς η κάθε κατάσταση διαρκεί δέκα λεπτά. Η επιλογή αυτή έγινε για να διατηρηθεί ισορροπία μεταξύ της ακρίβειας στην προσομοίωση και τον αριθμό των υπολογισμών που απαιτούνται για την εκτέλεση. Σε περίπτωση που απαιτηθεί, είναι εφικτό να γίνει αλλαγή της ανάλυσης του μοντέλου για να καλύψει πιθανές νέες μελλοντικές απαιτήσεις.

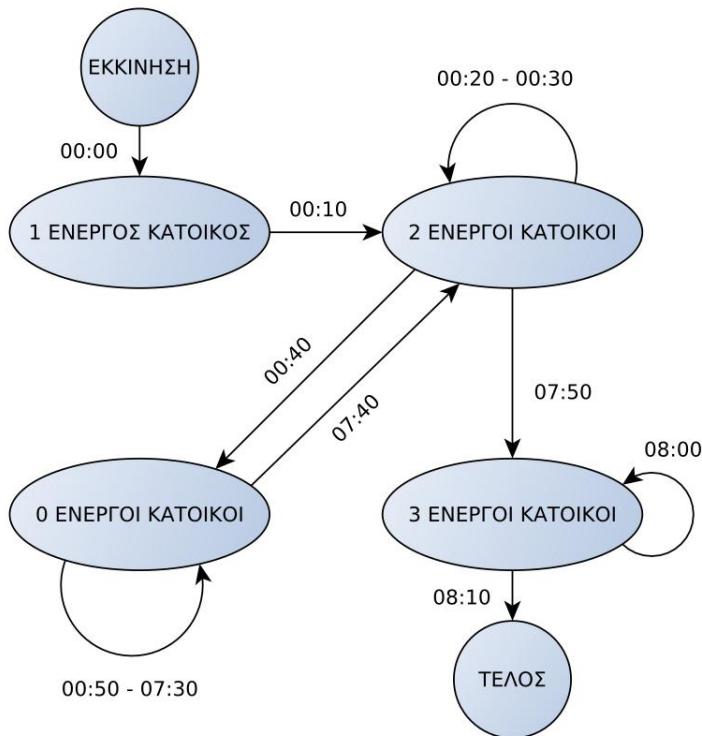
Το βασικότερο σημείο του μοντέλου είναι η ισχυρή εξάρτηση της οικιακής ζήτησης ενέργειας από τη δραστηριότητα και τις ανάγκες των κατοίκων του κτιρίου. Συνεπώς απαιτείται η μοντελοποίηση αυτών, καθώς και των παραγόντων που οδηγούν σε αυτές. Η ανάλυση αυτών ακολουθεί.

2.2. Ενεργοί Κάτοικοι

Οι “ενεργοί κάτοικοι” ορίζονται οι κάτοικοι που βρίσκονται σε θέση να έχουν κάποια δραστηριότητα η οποία επηρεάζει την ηλεκτρική κατανάλωση εντός του κτιρίου. Για τις ανάγκες του μοντέλου, αυτοί είναι οι κάτοικοι που βρίσκονται στο σπίτι και δεν κοιμούνται την εξεταζόμενη χρονική στιγμή. Παράγοντες που επηρεάζουν τον αριθμό των ενεργών κατοίκων είναι ο συνολικός αριθμός κατοίκων του κτιρίου, η ώρα της ημέρας, ο τύπος της ημέρας (αν είναι καθημερινή ή αργία), καθώς και ο αριθμός των ενεργών κατοίκων στο παρελθόν (την προηγούμενη χρονική στιγμή).

Η εξέλιξη του αριθμού των ενεργών κατοίκων προσομοιώνεται ως μια Μαρκοβιανή αλυσίδα, της οποίας οι πιθανές καταστάσεις είναι όλες οι πιθανές τιμές που μπορεί να έχει ο αριθμός των ενεργών κατοίκων, δηλαδή από 0 έως τον αριθμό των κατοίκων του κτιρίου κατά μέγιστο. Η κάθε κατάσταση διαρκεί 10 λεπτά, δηλαδή όσο ο χρόνος ανάλυσης που επιλέχθηκε για το μοντέλο.

Σχ. 2.1. Παράδειγμα Μαρκοβιανής αλυσίδας ενεργών κατοίκων



Όπως γίνεται σαφές από το Σχ. 2.1, ο προσδιορισμός των μεταβάσεων από κατάσταση σε κατάσταση αποτελεί το σημαντικότερο ζήτημα για την προσομοίωση. Αυτό συμβαίνει γιατί δεν είναι δυνατό να βρεθεί η αλυσίδα αν δε βρεθούν οι μεταβάσεις μεταξύ των καταστάσεων που την καθορίζουν. Καθώς οι μεταβάσεις αυτές είναι στοχαστικές, δε μπορούν να βρεθούν επακριβώς, αλλά μπορούν κάθε φορά να προσδιοριστούν οι πιθανότητες μετάβασης από μια κατάσταση σε κάθε άλλη.

Για τον προσδιορισμό αυτό χρησιμοποιείται μια πιθανοτική μέθοδος ως εξής: Όπως έχει αναφερθεί, υπάρχουν ανεξάρτητοι μεταξύ τους παράγοντες οι οποίοι επηρεάζουν τον αριθμό των ενεργών κατοίκων. Το γεγονός αυτό απεικονίζεται με πίνακες οι οποίοι προσδιορίζουν τις πιθανότητες μετάβασης από κατάσταση σε κατάσταση ανάλογα με τον κάθε ανεξάρτητο παράγοντα. Έτσι θα έχουμε πίνακες, όπως για παράδειγμα αυτόν του Σχ. 2.2. στον οποίο φαίνονται οι πιθανότητες μετάβασης σε καθεμιά από τις 6 καταστάσεις ενεργών κατοίκων (0 έως 5 ενεργοί κάτοικοι) ανάλογα με την ώρα της ημέρας, δηλαδή το συγκεκριμένο δεκάλεπτο από τα 144 που αντιστοιχούν σε 24 ώρες. Ομοίως υπάρχουν πίνακες που δείχνουν τον τρόπο που ο αριθμός των ενεργών κατοίκων στο προηγούμενο δεκάλεπτο (Σχ. 2.3), καθώς και ο συνολικός αριθμός των κατοίκων του κτιρίου και το είδος της ημέρας (καθημερινή ή αργία) επηρεάζουν τις πιθανότητες μετάβασης.

Στη συνέχεια με πολλαπλασιασμό αυτών των πινάκων και αναγωγή των τιμών στην τιμή 1, παράγεται ένας συνολικός πίνακας (Σχ. 2.6) που απεικονίζει τις πιθανότητες μετάβασης δεδομένου οποιουδήποτε συνδυασμού αυτών των ανεξαρτήτων παραγόντων. Ο πίνακας αυτός ονομάζεται “μήτρα μεταβάσεων”. Τελικά, κάνοντας χρήση αυτού του πίνακα σε συνδυασμό με μια γεννήτρια ψευδοτυχαίων αριθμών μπορεί να προσομοιωθεί με στατιστικά ορθό τρόπο η εξέλιξη των μεταβάσεων από κατάσταση σε κατάσταση.

Επειδή το μοντέλο βασίζεται, ανάμεσα στα άλλα, στη γνώση του αριθμού ενεργών κατοίκων στο προηγούμενο χρονικό διάστημα, ο οποίος στην αρχή είναι άγνωστος, είναι απαραίτητη η εκκίνηση της προσομοίωσης με μια αρχική τιμή. Επιλέγεται πάντοτε η αρχή της ημέρας να είναι στις 24:00, συνεπώς το πρώτο δεκάλεπτο θα είναι το 00:00 – 00:10. Η τιμή των ενεργών κατοίκων ακριβώς πριν από αυτό το πρώτο δεκάλεπτο βασίζεται στο συνδυασμένο πίνακα μεταβασης καθημερινής/αργίας και συνολικού αριθμού κατοίκων του κτιρίου (και τα δύο δεδομένης της χρονικής περιόδου) (Σχ. 2.4.). Επειδή ο πίνακας αυτός όπως χρησιμοποιείται ως έχει αλλά και σε συνδυασμό με τους άλλους, δεν κρίθηκε σκόπιμο να αναλυθεί σε δύο ανεξάρτητους πίνακες, αν και η διάσπαση αυτή είναι δυνατόν να γίνει στο μέλλον.

Οι τιμές των αρχικών πινάκων είναι κατά βάση ενδεικτικές επειδή δεν υπάρχουν στοιχεία διαθέσιμα για την Ελληνική πραγματικότητα, βασίζονται όμως σε υπάρχουσα μελέτη [20] η οποία δίνει τα αντίστοιχα στοιχεία για τη Μ. Βρετανία. Από τη μελέτη αυτή ο πίνακας που παρουσιάζεται στο Σχ. 2.4 χρησιμοποιείται αυτούσιος, καθώς εκτιμήθηκε πως η πληροφορία που περιέχει είναι αντιπροσωπευτική, ενώ για τους υπόλοιπους πίνακες έγινε εκτίμηση έτσι ώστε να αντιπροσωπεύουν την Ελλάδα.

Η εκτίμηση αυτή θεωρείται πως δεν επηρεάζει την ακρίβεια του μοντέλου εφόσον τα αποτελέσματα που παράγονται είναι κοντά στα πραγματικά. Σε κάθε περίπτωση, σκοπός αυτής της εργασίας είναι η υλοποίηση του μοντέλου προσομοίωσης και όχι η εύρεση απόλυτα σωστών στοιχείων για μια συγκεκριμένη περιοχή, συνεπώς το γεγονός αυτό δεν αποτελεί πρόβλημα.

Επιπλέον, ο μέγιστος αριθμός των κατοίκων του κτιρίου μπορεί να είναι έως 5, διότι στα δεδομένα που χρησιμοποιήθηκαν ο αριθμός κατοικιών με 6 ή περισσότερους κατοίκους είναι πολύ μικρός, και επακολούθως δεν υπάρχουν αξιόπιστα στατιστικά δεδομένα για αυτές.

Σχ. 2.2. Παράδειγμα πίνακα μεταβάσεων (ως προς την ώρα της ημέρας)

time	0	1	2	3	4	5
1	0.23	0.19	0.17	0.15	0.14	0.12
2	0.23	0.19	0.17	0.15	0.14	0.12
3	0.23	0.19	0.17	0.15	0.14	0.12
4	0.23	0.19	0.17	0.15	0.14	0.12
5	0.23	0.19	0.17	0.15	0.14	0.12
6	0.41	0.14	0.13	0.12	0.11	0.09
7	0.41	0.14	0.13	0.12	0.11	0.09
8	0.41	0.14	0.13	0.12	0.11	0.09
9	0.41	0.14	0.13	0.12	0.11	0.09
...
...
144	0.16	0.19	0.19	0.17	0.15	0.14

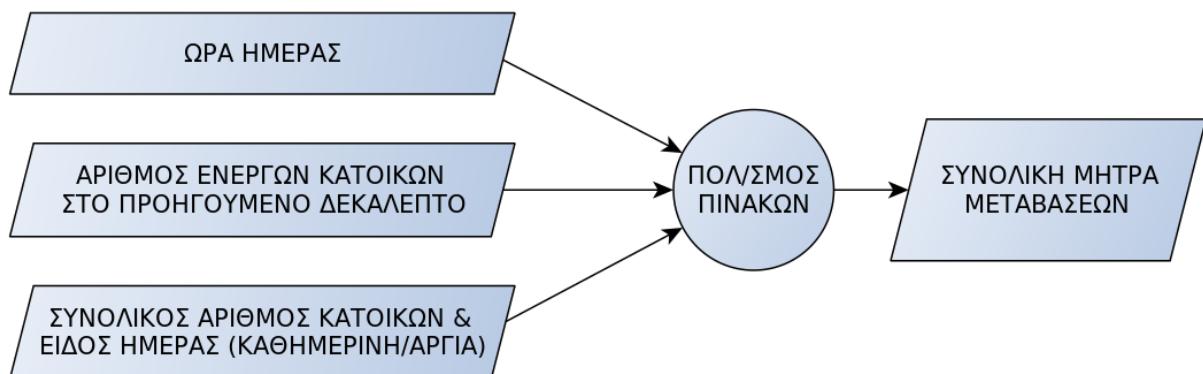
Σχ. 2.3. Παράδειγμα πίνακα μεταβάσεων (ως προς τον αριθμό ενεργών κατοίκων στο προηγούμενο δεκάλεπτο)

previous_active _occupants	0	1	2	3	4	5
0	0.41	0.22	0.175	0.095	0.075	0.025
1	0.22	0.38	0.19	0.095	0.07	0.045
2	0.12	0.2	0.355	0.165	0.095	0.065
3	0.1	0.14	0.24	0.32	0.12	0.08
4	0.09	0.12	0.15	0.21	0.31	0.12
5	0.08	0.13	0.16	0.18	0.2	0.25

Σχ. 2.4. Παράδειγμα πίνακα μεταβάσεων (ως προς τον συνολικό αριθμό κατοίκων του κτιρίου και το είδος της ημέρας – χρήση και για την εύρεση του αρχικού αριθμού κατοίκων)

weekend	residents	0	1	2	3	4	5
0	1	0.843	0.156	0	0	0	0
0	2	0.797	0.144	0.057	0	0	0
0	3	0.754	0.189	0.043	0.013	0	0
0	4	0.734	0.181	0.061	0.020	0.002	0
0	5	0.68	0.176	0.12	0.008	0.008	0.008
1	1	0.828	0.171	0	0	0	0
1	2	0.767	0.161	0.071	0	0	0
1	3	0.676	0.210	0.094	0.0188	0	0
1	4	0.675	0.207	0.092	0.015	0.009	0
1	5	0.546	0.25	0.109	0.078	0	0.015

Σχ. 2.5. Κατασκευή συνολικής μήτρας μεταβάσεων



Σχ. 2.6. Συνολική μήτρα μεταβάσεων

weekend	residents	previous_active_occupants	time	0	1	2	3	4	5
0	1	0	1	0.9239	0.0760	0	0	0	0
0	1	0	2	0.9239	0.0760	0	0	0	0
...
1	4	4	144	0.4794	0.3055	0.1609	0.0338	0.0202	0
1	4	4	145	0.2838	0.4256	0.2241	0.0423	0.0239	0
1	4	5	1
...
1	5	5	145

2.3. Συσκευές

Οι συσκευή ορίζεται κάθε ηλεκτρική συσκευή που μπορεί να συνεισφέρει στη ζήτηση ενέργειας από μια κατοικία. Οι συσκευές έχουν χωριστεί σε κατηγορίες (πχ. ψυγεία, πλυντήρια, συσκευές μαγειρέματος κλπ.) για να απλουστευτεί η περεταίρω ανάλυση. Κάθε συσκευή έχει χαρακτηριστικά που την προσδιορίζουν όπως για παράδειγμα ο τύπος της (είδος δραστηριότητας που καλύπτει), η ζήτηση ισχύος σε κατάσταση λειτουργίας, η ζήτηση ισχύος σε κατάσταση αδρανείας, η διάρκεια του μέσου κύκλου λειτουργίας και αδρανείας, αν η λειτουργία συσχετίζεται με ανθρώπινες δραστηριότητες, η πιθανότητα κατοχής της κ.α. Για συσκευές των οποίων η ζήτηση ενέργειας διαφέρει σημαντικά ανάλογα με το σημείο του κύκλου στον οποίο βρίσκονται (πχ. πλυντήριο), τα δεδομένα ζήτησης ενέργειας σε κάθε σημείο του κύκλου λειτουργίας δίνονται επίσης και είναι βασισμένα σε πραγματικές τυπικές συσκευές που κυκλοφορούν στο εμπόριο. Τέλος, η ζήτηση ενέργειας κάθε συσκευής πολλαπλασιάζεται με ένα ρυθμιστικό συντελεστή με σκοπό τη διόρθωση των τελικών αποτελεσμάτων αν απαιτείται (calibration).

Αρχικά, σε κάθε κτίριο που προσομοιώνεται, εκχωρείται ένας αριθμός συσκευών με πιθανοτική μέθοδο που βασίζεται στις πιθανότητες κατοχής κάθε συσκευής. Όπως αναφέρθηκε, κάθε κτίριο έχει ορισμένη πιθανότητα κατοχής μιας συγκεκριμένης συσκευής, συνεπώς αρκεί να χρησιμοποιηθεί μια γεννήτρια ψευδοτυχαίων αριθμών για επιλεγεί ένας ορισμένος συνδυασμός συσκευών για το κτίριο.

Η κατάσταση λειτουργίας όλων των συσκευών του κτιρίου μπορεί να αναπαρασταθεί ως Μαρκοβιανή αλυσίδα, της οποίας η κάθε κατάσταση αντιστοιχεί σε ένα συγκεκριμένο συνδυασμό καταστάσεων των συσκευών, συνεπώς σε μια συγκεκριμένη γενική ζήτηση ισχύος από τις συσκευές. Ομοίως η κατάσταση λειτουργίας κάθε συσκευής και αυτή αναπαρίσταται ως Μαρκοβιανή αλυσίδα με κάθε κατάσταση να αντιστοιχεί σε ορισμένη ζήτηση ισχύος της συσκευής. Οι μεταβάσεις μεταξύ των καταστάσεων μιας συσκευής εξαρτώνται από τα χαρακτηριστικά των συσκευών, καθώς και μια “μήτρα δραστηριοτήτων” (Σχ.2.7). Η μήτρα αυτή είναι ένας πίνακας ο οποίος περιέχει την πιθανότητα να εκτελεστεί κάποια δραστηριότητα (από ένα σύνολο δραστηριοτήτων), με δεδομένα την ώρα της ημέρας, τον αριθμό των ενεργών κατοίκων και το είδος της ημέρας (αργία ή καθημερινή). Ο πίνακας αυτός σχηματίστηκε βάσει των στατιστικών δεδομένων συμπεριφοράς που βρίσκονταν στη

διάθεση της εργασίας αυτής. Για να εκτελεστούν αυτές οι δραστηριότητες απαιτείται η χρήση μιας αντίστοιχης συσκευής (επιλεγμένης βάσει του τύπου της για να καλύψει την ανάγκη που προέκυψε). Για παράδειγμα, αν η δραστηριότητα είναι το μαγείρεμα, ο τύπος συσκευών που απαιτείται να ενεργοποιηθεί είναι οι συσκευές μαγειρέματος (πχ. φούρνος, μάτι κουζίνας, τοστιέρα κλπ).

Η προσομοίωση εκτελείται για κάθε δεκάλεπτο μιας ημέρας και κατά την εκτέλεση ελέγχεται κάθε συσκευή του κτιρίου για να βρεθεί η κατάσταση ζήτησης ενέργειάς της στο επόμενο δεκάλεπτο, δηλαδή ουσιαστικά αν υπάρχει γεγονός έναυσης ή σβέσης.

Αν η συσκευή βρισκόταν σε κατάσταση σβέσης, γίνεται έλεγχος για το αν θα υπάρξει έναυση. Συσκευές που εξαρτώνται από την ανθρώπινη δραστηριότητα (πχ. μαγείρεμα) εκκινούν αν ενεργοποιηθεί η δραστηριότητα που απαιτεί τη χρήση τους και λειτουργούν μέχρι να ολοκληρωθεί ο κύκλος τους (άρα και η δραστηριότητα). Αν στο μεταξύ ο αριθμός των ενεργών κατοίκων μηδενιστεί, ανάλογα και με τον τύπο της συσκευής, θα έχουμε σβέση και η δραστηριότητα θα ολοκληρωθεί όταν υπάρξει ξανά τουλάχιστον ένας ενεργός κάτοικος. Αντίθετα, συσκευές που η λειτουργία τους δεν εξαρτάται από την ανθρώπινη δραστηριότητα (πχ. ψυγείο) εκκινούν όταν ο κύκλος αδρανείας τους τελειώσει και λειτουργούν όσο ορίζεται από τον κύκλο λειτουργίας τους. Σε συσκευές που λειτουργούν εποχιακά, όπως θερμοπομποί που λειτουργούν όλο το χειμώνα, προσομοιώνονται στοχαστικά τα γεγονότα έναυσης & σβέσης, και για το διάστημα που είναι ενεργές (μερικούς μήνες) η ζήτηση ενέργειάς τους βασίζεται στα χαρακτηριστικά των κύκλων λειτουργίας/αδρανείας τους.

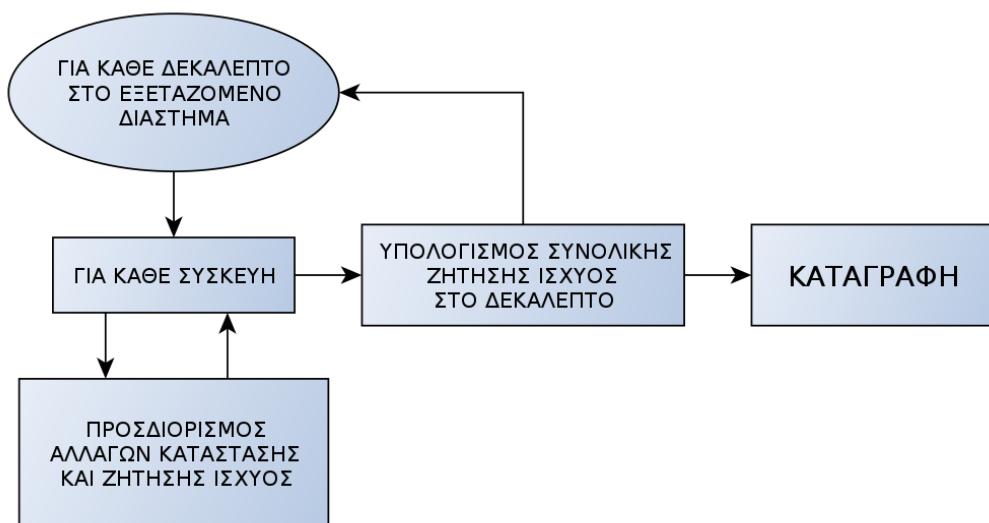
Στα δεδομένα μας περιλαμβάνονται και συσκευές των οποίων ο κύκλος λειτουργίας διαρκεί λιγότερο από δέκα λεπτά, ή των οποίων η διάρκεια του κύκλου δεν αποτελεί ακέραιο πολλαπλάσιο του δέκα. Σε αυτή την περίπτωση δε μπορούμε να χρησιμοποιήσουμε την τιμή ζήτησης ισχύος σε Watt ως έχει, καθώς θα αναπαριστούσε ζήτηση ισχύος για ένα ολόκληρο δεκάλεπτο, κάτι που δεν ισχύει. Γι αυτό το λόγο χρησιμοποιούμε γραμμική παρεμβολή για να αντιστοιχίσουμε τη ζήτηση ισχύος για το κλάσμα του δεκαλέπτου στο οποίο η συσκευή είναι ενεργή στη ζήτηση ισχύος της συσκευής στο εξεταζόμενο δεκάλεπτο.

Επιπλέον, επειδή η ζήτηση της κάθε συσκευής στην πραγματικότητα δεν είναι σταθερή, για συσκευές στις οποίες δεν παρέχεται συγκεκριμένη τιμή ζήτησης σε κάθε σημείο του κύκλου λειτουργίας, με κάθε έναυση ορίζεται μια νέα τιμή ζήτησης ισχύος που ισχύει για το σύνολο αυτού του κύκλου λειτουργίας. Η τιμή αυτή βρίσκεται ως τυχαίο σημείο μιας κανονικής κατανομής με μέσο την ονομαστική τιμή ζήτησης ισχύος της συσκευής και τυπική απόκλιση ίση με το 1/10 της αρχικής τιμής. Αντίστοιχα μεταβάλλεται και η διάρκεια του κύκλου λειτουργίας σε κάθε έναυση, ακολουθώντας κανονική κατανομή με μέσο τη γνωστή διάρκεια του κύκλου λειτουργίας όπως ορίζεται στα αρχικά δεδομένα και απόκλιση ίση με το 1/10 της τιμής αυτής.

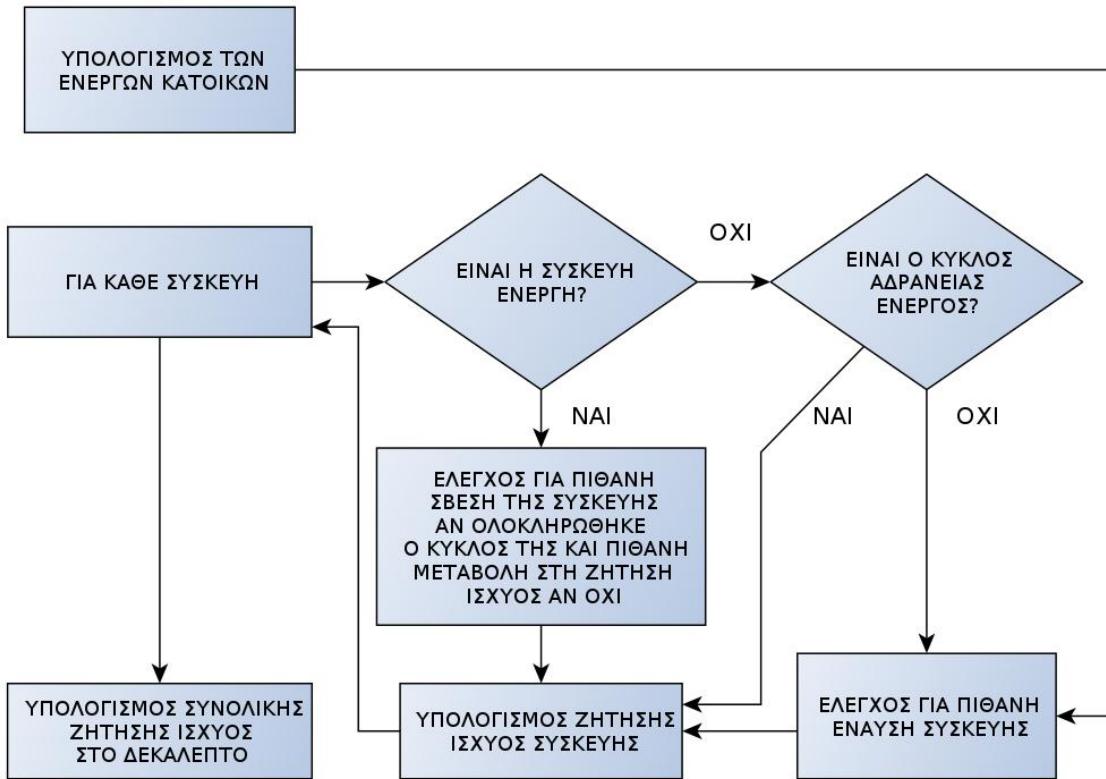
Σχ. 2.7. Μήτρα Δραστηριοτήτων [6]

weekend	active_occupants	activity	1	2	3	...	144
0	0	ACT_TV	0.000	0.000	0.000
0	0	ACT_COOKING	0.000	0.000	0.000
0	0	ACT_LAUNDRY	0.000	0.000	0.000
0	0	ACT_WASHDRESS	0.000	0.000	0.000
0	0	ACT_IRON	0.000	0.000	0.000
0	0	ACT_HOUSECLEAN	0.000	0.000	0.000
0	1	ACT_TV	0.322	0.329	0.336
0	1	ACT_COOKING	0.022	0.014	0.014
0	1	ACT_LAUNDRY	0.000	0.001	0.000
0	1	ACT_WASHDRESS	0.123	0.093	0.068
...
1	5	ACT_HOUSECLEAN

Σχ. 2.8. Μοντέλο ζήτησης ηλεκτρικής ισχύος από συσκευές



Σχ. 2.9. Διάγραμμα ροής προσδιορισμού ζήτησης ισχύος μιας συσκευής



2.4. Φωτισμός

Τα φωτιστικά σώματα, παρόλο και αυτά ανήκουν στην ευρύτερη κατηγορία των συσκευών, απαιτείται να εξεταστούν ξεχωριστά. Αυτό συμβαίνει γιατί ο τρόπος λειτουργίας τους διαφέρει αρκετά από αυτόν των υπολοίπων συσκευών. Η λειτουργία των φωτιστικών σωμάτων εξαρτάται τόσο από τον εξωτερικό φωτισμό όσο και από τη δραστηριότητα των κατοίκων. Επιπλέον το σημείο στο οποίο έχουν εγκατασταθεί παίζει σημαντικό ρόλο στο πόσο χρησιμοποιούνται και ο κύκλος λειτουργίας τους διαφέρει από αυτόν των άλλων συσκευών διότι σχετίζεται απευθείας με την ύπαρξη ή μη ενεργών κατοίκων στο δωμάτιο.

Σε κάθε κτίριο αρχικά εκχωρείται ένας αριθμός φωτιστικών με πιθανοτική μέθοδο. Συγκεκριμένα γίνεται τυχαία επιλογή ενός από 100 διαφορετικούς συνδυασμούς (τύπους κτηρίου), ο καθένας με συγκεκριμένο αριθμό, ισχύ και τεχνολογία φωτιστικών σωμάτων. Τα κτίρια αυτά έχουν παραχθεί βάσει στατιστικών δεδομένων και είναι αντιπροσωπευτικά πραγματικών κτιρίων.

Η λειτουργία των φωτιστικών μπορεί, παρόμοια με αυτή των συσκευών, να προσομοιωθεί ως Μαρκοβιανή αλυσίδα, όπου κάθε συνδυασμός καταστάσεων λειτουργίας των φωτιστικών (αναμμένα/σβηστά) αποτελεί μια Μαρκοβιανή κατάσταση. Με όμοιο τρόπο η λειτουργία του κάθε ξεχωριστού φωτιστικού προσομοιώνεται και αυτή ως Μαρκοβιανή αλυσίδα, με την κατάσταση σε κάθε χρονική στιγμή να αντιπροσωπεύει τη ζήτηση ενέργειας του φωτιστικού. Η μετάβαση από μια κατάσταση σε μια άλλη εξαρτάται από τον εξωτερικό φωτισμό, τον αριθμό των ενεργών κατοίκων και ένα συντελεστή “χρήσης” του φωτιστικού

που απεικονίζει τη διαφορετική χρήση που έχουν τα φωτιστικά ανάλογα με το σημείο που είναι εγκατεστημένα.

Ο εξωτερικός φωτισμός βασίζεται σε στατιστικά δεδομένα ανά λεπτό της ημέρας και ανά μήνα του έτους. Έχει γίνει η υπόθεση ότι ο εξωτερικός φωτισμός δεν αλλάζει χαρακτηριστικά στη διάρκεια ενός μήνα. Με βάση αυτά τα δεδομένα, και με επιλογή ενός επιπέδου εξωτερικού φωτισμού ως η βάση κάτω από την οποία απαιτείται τεχνητός φωτισμός, η προσομοίωση γίνεται για κάθε δεκάλεπτο της ημέρας. Ακόμα και αν ο εξωτερικό φωτισμός είναι επαρκής, πάντα υπάρχει η πιθανότητα να απαιτηθεί τεχνητός φωτισμός (πχ. σε δωμάτια χωρίς παράθυρα). Τελικά η απαίτηση ή μη τεχνητού φωτισμού είναι μια μεταβλητή αλήθειας (ναι/οχι).

Επιπλέον, λαμβάνεται υπόψη το φαινόμενο της ‘κοινής χρήσης φωτιστικών’, δηλαδή το φαινόμενο της τάσης για συνύπαρξη στο φωτισμένο δωμάτιο των κατοίκων του κτιρίου, χρησιμοποιώντας την έννοια των “πραγματικών ενεργών κατοίκων”, δηλαδή των ενεργών κατοίκων που έχουν ανάγκη ανεξάρτητο φωτισμό. Αυτός ο αριθμός προκύπτει από τον αριθμό των ενεργών κατοίκων βάσει ενός πίνακα αναγωγής και είναι μικρότερος ή ίσος με τον αριθμό ενεργών κατοίκων (Σχ. 2.10). Ο πίνακας αναγωγής που χρησιμοποιείται παρουσιάστηκε στην εργασία των Richardson I. et al. [7].

Για κάθε φωτιστικό σώμα και για κάθε δεκάλεπτο εξετάζουμε την κατάστασή του. Αν είναι σβηστό, βρίσκουμε την πιθανότητα έναυσης ως το γινόμενο της απαίτησης τεχνητού φωτισμού, του συντελεστή χρήσης του φωτιστικού, του αριθμού “πραγματικών ενεργών κατοίκων”, ενός ρυθμιστικού συντελεστή που εισάγεται με σκοπό την διόρθωση των αποτελεσμάτων και του απολύτου του λογαρίθμου ενός τυχαίου αριθμού μεταξύ 0 και 1, ο οποίος υπολογίζεται κατά τη δημιουργία του φωτιστικού και ο οποίος χρησιμεύει στην κατασκευή διαφορετικών προφίλ χρήσης για κάθε φωτιστικό.

Από τη στιγμή που υπάρχει γεγονός έναυσης, υπολογίζεται ο χρόνος λειτουργίας του φωτιστικού με βάση τον πίνακα χρόνου λειτουργίας που παρουσιάζεται στο Σχ. 2.11. Στον πίνακα αυτό έχουμε 9 διακριτές καταστάσεις, οι οποίες αντιστοιχούν σε ένα άνω και ένα κάτω όριο διάρκειας (πχ. για την κατάσταση (1), η διάρκεια λειτουργίας μπορεί να είναι 1' λεπτό, ενώ για την κατάσταση (6) η διάρκεια είναι από 17' έως 27' λεπτά). Γενικά ο χρόνος λειτουργίας είναι από λίγα λεπτά έως ~1,5 ώρα. Ο ακριβής αριθμός της διάρκειας λειτουργίας υπολογίζεται στοχαστικά ως τυχαίο σημείο κανονικής κατανομής ανάμεσα στα δύο άκρα (lower_value και upper_value).

Τέλος, σβέση ενός φωτιστικού έχουμε αν ολοκληρωθεί ο κύκλος λειτουργίας του, ή αν ο αριθμός των ενεργών κατοίκων γίνει 0, οπότε δεν απαιτείται πλέον φωτισμός.

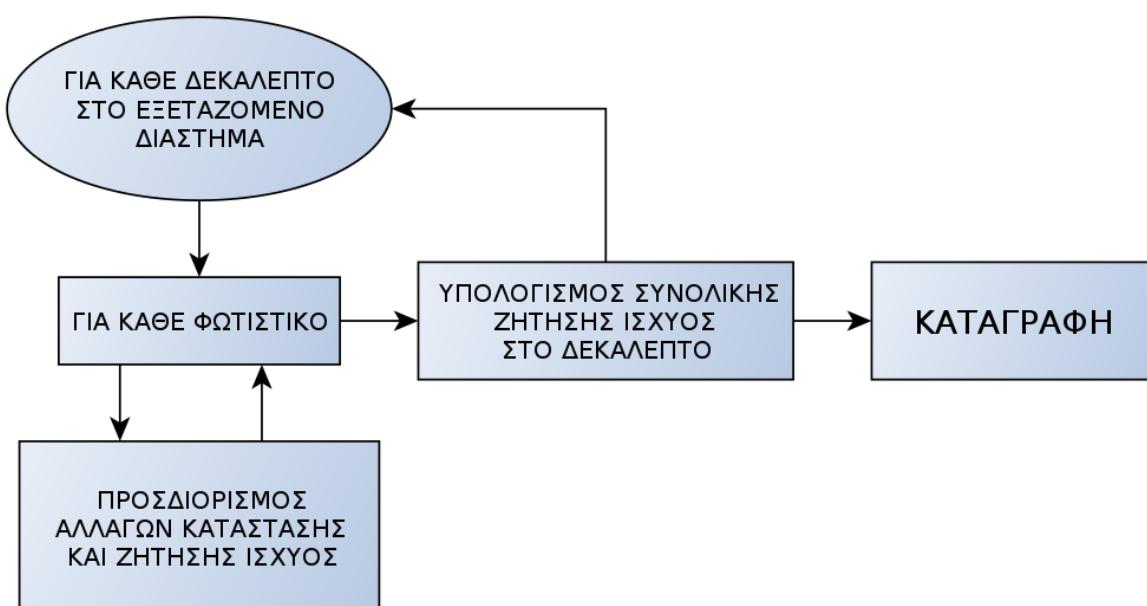
Σχ. 2.10. Πίνακας αναγωγής του αριθμού ενεργών κατοίκων σε αριθμό “πραγματικών ενεργών κατοίκων” [7]

active_occupants	0	1	2	3	4	5
effective_occupancy	0	1	1.528	1.694	1.983	2.094

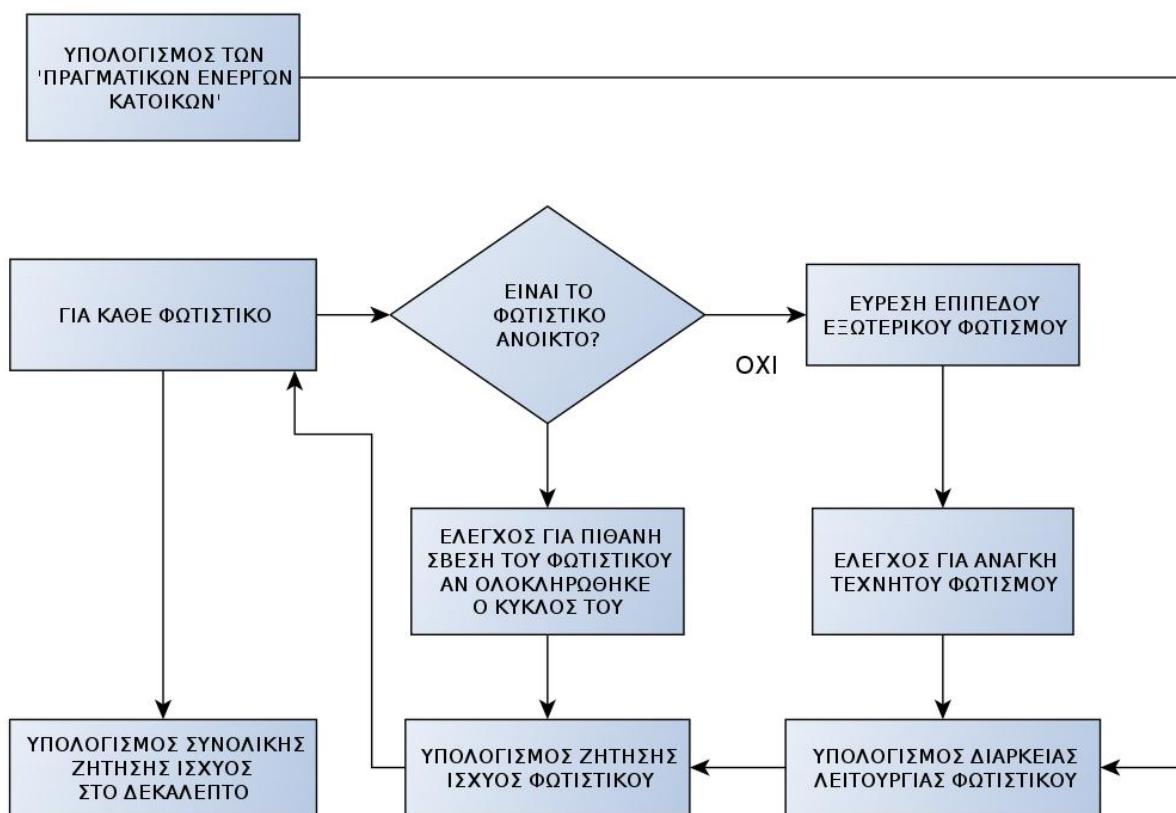
Σχ. 2.11. Χρόνος λειτουργίας φωτιστικών σωμάτων εφόσον γίνει έναυση [7]

range_of_equal_probability_number	lower_value	upper_value	cumulative_probability
1	1	1	0.111111
2	2	2	0.222222
3	3	4	0.333333
4	5	8	0.444444
5	9	16	0.555556
6	17	27	0.666667
7	28	49	0.777778
8	50	91	0.888889
9	92	259	1

Σχ. 2.12. Μοντέλο ζήτησης ηλεκτρικής ισχύος από φωτιστικά



Σχ. 2.13. Διάγραμμα ροής προσδιορισμού ζήτησης ισχύος ενός φωτιστικού



3. Υλοποίηση

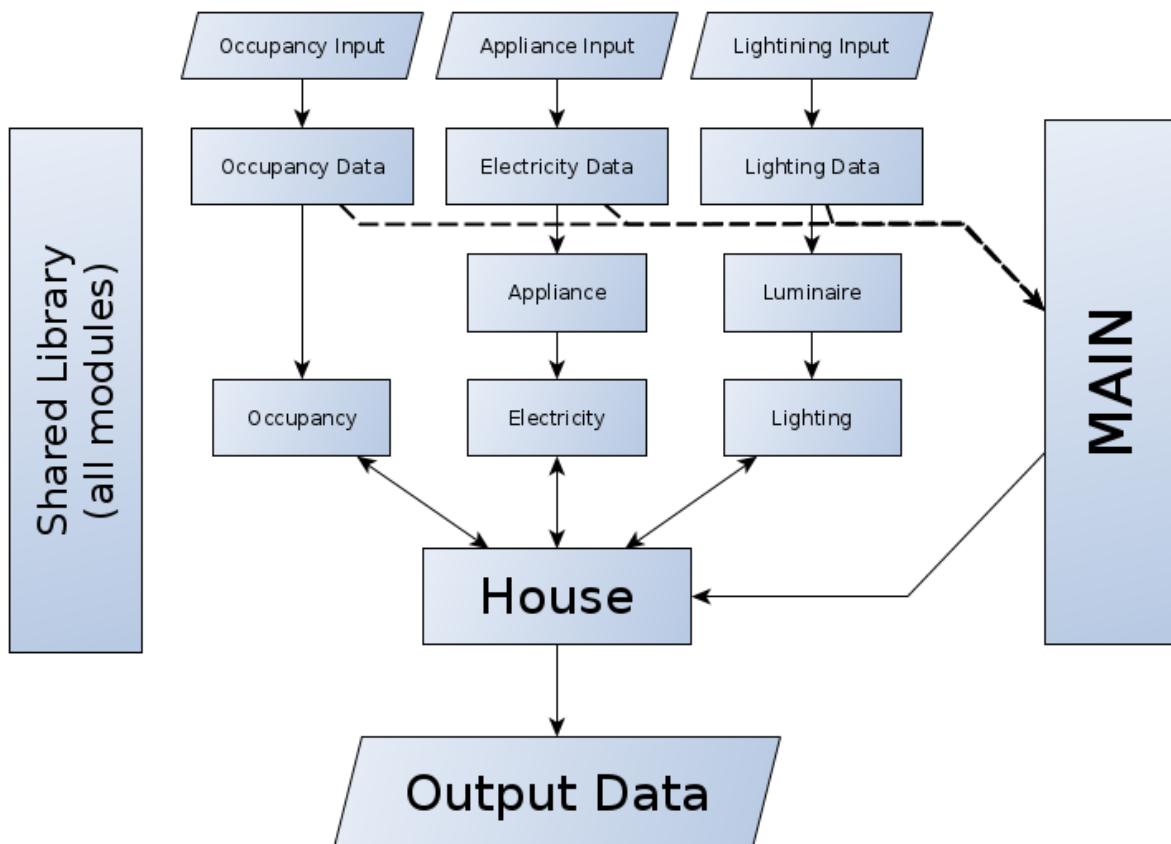
3.1. Γενικά

Η υλοποίηση του μοντέλου έγινε με χρήση της γλώσσας προγραμματισμού Python [21], χρησιμοποιώντας τεχνικές σπονδυλωτού (modular) [22] [23] και αντικειμενοστραφή (object-oriented) προγραμματισμού [22] [24] [25]. Η επιλογή αυτή έγινε με σκοπό την ταχεία ανάπτυξη της εφαρμογής, τηρώντας ωστόσο τεχνικές ορθού προγραμματισμού, έχοντας επίσης ως στόχο ικανοποιητικές ταχύτητες εκτέλεσης. Λόγω των επιλογών αυτών, η προσθήκη νέων παραμέτρων και λειτουργιών στην εφαρμογή είναι εύκολη και γρήγορη, πράγμα που αναμένεται να ωφελήσει και τη μελλοντική περεταίρω εξέλιξή της. Για την εξασφάλιση καλής ταχύτητα εκτέλεσης αλλά και λαμβάνοντας υπόψη τη δυνατότητα εύρεσης δεδομένων, επιλέχθηκε η ανάλυση των δεδομένων στην εφαρμογή να είναι ενός δεκαλέπτου. Ως αποτέλεσμα αυτού, τα παραγόμενα δεδομένα είναι αρκετά υψηλής ανάλυσης, ενώ μπορούμε εύκολα με άθροισή τους να παράγουμε δεδομένα χαμηλότερης ανάλυσης αν απαιτηθεί από την εφαρμογή. Δεδομένα εισάγονται στην εφαρμογή από αρχεία μορφής .csv η οποία επιλέχθηκε τόσο λόγω της καλής συμβατότητάς της με άλλα προγράμματα όσο και της απλότητας και ευχέρειας που δίνει στην επεξεργασία των δεδομένων. Τα παραγόμενα δεδομένα επιλέχθηκε να εξάγονται σε μορφή .xlsx (φύλλου excel), και πάλι λόγω της καλής συμβατότητας της μορφής αυτής με άλλες εφαρμογές, αλλά και της ευκολίας στην οπτικοποίηση και περεταίρω επεξεργασία των δεδομένων. Στην εφαρμογή έγινε εκτεταμένη χρήση των βιβλιοθηκών της python ‘numpy’ και ‘pandas’ διότι οι βιβλιοθήκες αυτές ενδείκνυνται για την ταχεία επεξεργασία και ανάλυση του μεγάλου όγκου δεδομένων που εισάγονται αλλά και παράγονται από την εφαρμογή. Πιο συγκεκριμένα:

- **Numpy:** Η βιβλιοθήκη αυτή προσθέτει μια σειρά από αποδοτικές μαθηματικές συναρτήσεις και εργαλεία ανάλυσης όπως για παράδειγμα, N-διάστατους πίνακες, μεθόδους γραμμικής άλγεβρας, γεννήτριες ψευδοτυχαίων αριθμών κ.α. [26]. Συγκεκριμένα στην εφαρμογή χρησιμοποιούνται οι μαθηματικές συναρτήσεις για κατασκευή μιας σειράς από πίνακες, πολλαπλασιασμό πινάκων, εύρεση τυχαίων αριθμών βάσει τυχαίων ή κανονικών κατανομών κ.α.
- **Pandas (Python Data Analysis Library):** Η βιβλιοθήκη αυτή δίνει δυνατότητες χρήσης δομών δεδομένων και εργαλείων ανάλυσης υψηλής απόδοσης. Οι δομές αυτές μπορούν να είναι ιεραρχικές, ενώ πράξεις πάνω σε δομές pandas έχουν πολύ καλή απόδοση. Επιπλέον παρέχονται δυνατότητες λειτουργιών I/O προς μια σειρά διαφορετικών τύπων αρχείων, όπως τα .csv και .xlsx που χρησιμοποιήθηκαν στην υλοποίηση αυτή [27]. Συγκεκριμένα στην εφαρμογή γίνεται χρήση δομών DataFrame οι οποίες στην πράξη αποτελούν πίνακες τύπου numpy, συμπεριλαμβάνοντας όμως μεταδεδομένα και συγκεκριμένη ιεραρχία βάσει των οποίων μπορούν να επιλεγούν τμήματα των δομών αυτών με τρόπο γρήγορο, εύκολο και κατανοητό, καθώς και να εκτελεστούν πράξεις ταχύτατα σε όλο τον πίνακα ή σε μέρος αυτού. Οι δομές αυτές χρησιμοποιούνται τόσο για την εισαγωγή και χρήση σχεδόν όλων των δεδομένων από τα αρχεία .csv όσο και για την αποθήκευση και επεξεργασία των παραγόμενων δεδομένων της εφαρμογής.

Στη συνέχεια ακολουθεί περιγραφή των τμημάτων της εφαρμογής. Το καθένα από αυτά αποτελεί μια αυτοτελή κλάση στην οποία εμπεριέχονται όλες οι λειτουργίες του τμήματος, εκτός αν αναφέρεται διαφορετικά.

Σχ. 3.1. Διαγραμματική απεικόνιση εφαρμογής:



3.2. Κύρια Κλάση (main)

Η ‘κύρια’ κλάση είναι η ανώτατη κλάση της εφαρμογής. Περιλαμβάνει τις θέσεις που βρίσκονται τα απαραίτητα δεδομένα για την εφαρμογή, ενώ ρόλος της είναι γενικά ο έλεγχος όλης της εφαρμογής.

Πιο συγκεκριμένα η κύρια κλάση εκτελεί τις εξής λειτουργίες κατά την εκκίνησή της:

- Επαναδημιουργεί τμήματα των απαραιτήτων δεδομένων, αν τα δεδομένα στα οποία αυτά βασίζονται μεταβληθούν
- Κατασκευάζει τις μικρότερου επιπέδου κλάσεις που είναι απαραίτητες για τη λειτουργία της εφαρμογής (με χρήση των αρχικών δεδομένων που φορτώνονται στη μνήμη με την εκκίνηση της κλάσης)
- Εκτελεί το μοντέλο και αποθηκεύει τα δεδομένα που παράχθηκαν με χρήση της μεθόδου “run_model”. Η μέθοδος αυτή χρησιμοποιεί αρχικά τη μέθοδο

“create_house” για να κατασκευάσει όσες ανεξάρτητες κλάσεις τύπου ‘κτίριο’ είναι απαραίτητες για την προσομοίωση. Στη συνέχεια για κάθε κτίριο εκτελεί τη μέθοδο προσομοίωσης για το ζητούμενο χρονικό διάστημα και αποθηκεύει τα αποτελέσματα για κάθε κτίριο ξεχωριστά.

3.3. Κτίριο (house)

Η κλάση ‘κτίριο’ είναι μάλλον η σημαντικότερη κλάση της εφαρμογής, καθώς εμπεριέχει τα στοιχεία κάθε ανεξάρτητου κτιρίου που εξετάζεται. Περιλαμβάνει τα εξής χαρακτηριστικά (attributes) που δίνονται ως είσοδοι κατά την εκκίνησή:

- Αριθμός μονίμων κατοίκων του κτιρίου
- Συνδυασμός συσκευών του κτιρίου
- Συνδυασμός φωτιστικών σωμάτων του κτιρίου
- Όριο του εξωτερικού φωτισμού κάτω από το οποίο ο τεχνητός φωτισμός κρίνεται απαραίτητος

Ο ρόλος της είναι η εκτέλεση της πλήρους διεργασίας προσομοίωσης για το κτίριο εκτελώντας τα απαραίτητα υποτυμήματα αυτής δηλαδή:

- Εκτέλεση προσομοίωσης και παραγωγή δεδομένων ενεργών κατοίκων για το χρονικό διάστημα που απαιτείται (μέθοδος “get_occupancy”)
- Εκτέλεση προσομοίωσης και παραγωγή δεδομένων ζήτησης ηλεκτρικής ενέργειας των συσκευών για το χρονικό διάστημα που απαιτείται, με χρήση των δεδομένων ενεργών κατοίκων (μέθοδος “get_appliance_demand”)
- Εκτέλεση προσομοίωσης και παραγωγή δεδομένων ζήτησης ηλεκτρικής ενέργειας των φωτιστικών σωμάτων για το χρονικό διάστημα που ζητείται, με χρήση των δεδομένων ενεργών κατοίκων (μέθοδος “get_luminaire_demand”)
- Επεξεργασία και σύνθεση όλων των ανωτέρω δεδομένων και ακόλουθα παραγωγή των συνολικών δεδομένων ζήτησης ενέργειας στο χρονικό διάστημα που μας ενδιαφέρει, σε μορφή ιεραρχικής (multiindex) δομής πίνακα τύπου Pandas DataFrame όπως απαιτούν οι προδιαγραφές της εφαρμογής. (μέθοδος “get_power_demand”)

3.4. Δεδομένα Συσκευών (appliance_data)

Η κλάση αυτή περιλαμβάνει μεθόδους που επιστρέφουν έναν ορισμένο συνδυασμό συσκευών για κάθε κτίριο. Χρησιμοποιείται κατά τη δημιουργία του κτιρίου από την κλάση “κτίριο”. Κατά την εκκίνησή της δίνονται δεδομένα σχετικά με τις διαθέσιμες συσκευές και τα χαρακτηριστικά τους. Με τη μέθοδο “assign_new_appliances” δημιουργείται ο συνδυασμός των συσκευών που ανήκουν στο κτίριο βάσει των πιθανοτήτων κατοχής του τύπου των συσκευών αυτών που λαμβάνονται από τα χαρακτηριστικά τους. Στη συνέχεια δημιουργούνται αυτές οι συσκευές ως μέλη της κλάσης “συσκευή” και επιστρέφονται αποτελώντας στοιχεία της κλάσης “κτίριο”.

3.5. Δεδομένα Φωτιστικών (luminaire_data)

Η κλάση αυτή περιλαμβάνει μεθόδους που επιστρέφουν έναν ορισμένο συνδυασμό φωτιστικών για κάθε κτίριο. Χρησιμοποιείται κατά τη δημιουργία του κτιρίου από την κλάση “κτίριο”. Κατά την εκκίνησή της δίνονται δεδομένα σχετικά με τους διαθέσιμους συνδυασμούς φωτιστικών και τα χαρακτηριστικά τους. Με τη μέθοδο “assign_new_luminaires” επιλέγεται ένας ορισμένος συνδυασμός φωτιστικών βάσει των πιθανοτήτων εμφάνισης του συνδυασμού αυτού σε ένα κτίριο. Στη συνέχεια δημιουργούνται τα φωτιστικά σώματα ως μέλη της κλάσης “φωτιστικό σώμα” και επιστρέφονται αποτελώντας στοιχεία της κλάσης “κτίριο”. Τέλος, μέσω της μεθόδου “get_house_irradiance_threshold”, ορίζεται το κατώτατο όριο εξωτερικού φωτισμού πέραν του οποίου απαιτείται για τη χρήση των φωτιστικών του κτιρίου, ως αποτέλεσμα κανονικής κατανομής με μέσο και τυπική απόκλιση ορισμένα από τα αρχικά δεδομένα.

3.6. Δεδομένα Ενεργών Κατοίκων (occupancy_data)

Η κλάση αυτή κατά την εκκίνηση παίρνει ως είσοδο τις διευθύνσεις των δεδομένων που αφορούν τους ενεργούς κατοίκους δηλαδή:

- Της αρχικής μήτρας μετάβασης για συσκευές
- Της τελικής μήτρας μετάβασης για συσκευές
- Του αρχείου που περιέχει τις πιθανότητες μετάβασης σε διαφορετική κατάσταση ενεργών κατοίκων ως προς τους διάφορους παράγοντες επηρεάζουν τη μετάβαση

Ο ρόλος της είναι σχετικός με τη φόρτωση αλλά και την πιθανή επαναδημιουργία της τελικής μήτρας μεταβάσεων.

Σχετικά με την επαναδημιουργία (μέθοδος create_general_transitional_matrix):

- Λαμβάνονται ως βάση όλοι οι πίνακες μετάβασης ως προς τους παράγοντες που μας αφορούν
- Ανιχνεύεται ο αριθμός και η θέση των πινάκων που πρέπει να ληφθούν υπόψη και φορτώνονται στη μνήμη

Ανά δύο ακολουθείται η εξής διαδικασία που σκοπό έχει τελικά την κατασκευή της συνολικής μήτρας μεταβάσεων:

- Ο πρώτος πίνακας επεκτείνεται τόσες φορές όσες και οι γραμμές του δευτέρου. Δημιουργείται δηλαδή ένας νέος πρώτος πίνακας που αποτελείται από τόσες επαναλήψεις του αρχικού πρώτου, όσες και οι γραμμές του δευτέρου πίνακα
- Κάθε γραμμή του δευτέρου πίνακα πολλαπλασιάζεται με κάθε γραμμή της επανάληψης του πρώτου πίνακα που της αντιστοιχεί, συγκεκριμένα πολλαπλασιάζοντας κάθε στοιχείο της γραμμής αυτής με τα αντίστοιχά τους στο νέο πίνακα (αντίστοιχα θεωρούνται τα στοιχεία που έχουν τον ίδιο αριθμό στήλης)
- Στο τέλος ο νέος πίνακας αποτελεί συνδυασμό όλων των γραμμών των δύο πινάκων από τους οποίους προήλθε, έχοντας την επιθυμητή σειρά στα δεδομένα που περιέχει

Επαναλαμβάνοντας τη διαδικασία αυτή για κάθε πίνακα μεταβάσεων που δόθηκε ως βάση, καταλήγουμε τελικά στη συνολική μήτρα μεταβάσεων.

Τελικά, ανάλογα και με τα μεταδεδομένα των αρχικών πινάκων μεταβάσεων, κατασκευάζεται το ευρετήριο του τελικού πίνακα μεταβάσεων το οποίο είναι της μορφής "(weekend, residents, previous_active_occupants, time)", δηλαδή τα δεδομένα που περιέχονται στο ευρετήριο υποδηλώνουν την κατάσταση όλων αυτών των παραγόντων στην εκάστοτε γραμμή.

Ο συνδυασμός του ευρετηρίου με τον πίνακα είναι μια δομή τύπου pandas DataFrame, από την οποία η εύρεση των κάθε φορά απαιτούμενων δεδομένων γίνεται εύκολα και γρήγορα. Αυτή αποθηκεύεται στο δίσκο με σκοπό τη μελλοντική χρήση για όσο τα βασικά δεδομένα δε μεταβάλλονται.

Αν δε γίνει επαναδημιουργία, η αποθήκευμένη μήτρα απλά φορτώνεται ως έχει, καθώς είναι έτοιμη προς χρήση. Η τελική μήτρα μεταβάσεων, είτε κατασκευάστηκε εκ νέου, είτε φορτώθηκε από το δίσκο, είναι στη διάθεση των άλλων κλάσεων, και ιδιαίτερη χρήση της γίνεται από την κλάση 'Ενεργοί Κάτοικοι'

3.7. Ενεργοί Κάτοικοι (occupancy)

Η κλάση αυτή περιλαμβάνει μεθόδους οι οποίες σε ένα ορισμένο χρονικό διάστημα, για κάθε ημέρα και για κάθε δεκάλεπτο κάθε ημέρας αυτού, δημιουργούν δεδομένα ενεργών κατοίκων, χρησιμοποιώντας ως εισόδους τις μήτρες μεταβάσεων από τη μια κατάσταση στην άλλη, καθώς και τα στοιχεία που από τα οποία εξαρτάται η μετάβαση (μέθοδος "occupancy_generator")

Πιο συγκεκριμένα, για την παραγωγή στοιχείων ενεργών κατοίκων για μια συγκεκριμένη στιγμή της ημέρας, από τη μήτρα μεταβάσεων βρίσκεται η πιθανότητα μετάβασης σε κάθε πιθανή κατάσταση βάσει των στοιχείων από τα οποία εξαρτάται η μετάβαση:

- Συνολικοί κάτοικοι κτιρίου
- Ωρα της ημέρας
- Ενεργοί κάτοικοι το προηγούμενο δεκάλεπτο.
- Τύπος ημέρας (εργάσιμη ή αργία)

Στη συνέχεια, με χρήση γεννήτριας ψευδοτυχαίων αριθμών βρίσκεται η μετάβαση και μαζί της ο αριθμός των ενεργών κατοίκων για κάθε δεκάλεπτο του χρονικού διαστήματος που εξετάζεται (μέθοδοι "occupancy_data_updater" και "day_occupancy_generator")

Η προσομοίωση γίνεται σειριακά, διότι απαιτείται η γνώση του αριθμού των ενεργών κατοίκων στο προηγούμενο δεκάλεπτο για να βρούμε τον αντίστοιχο αριθμό στο τρέχον δεκάλεπτο. Όπως αναφέρθηκε και παραπάνω, κατά την εκκίνηση της προσομοίωσης ο αριθμός των ενεργών κατοίκων για το προηγούμενο δεκάλεπτο είναι άγνωστος, γι' αυτό για το δεκάλεπτο ακριβώς πριν την εκκίνηση χρησιμοποιείται η μήτρα 'αρχικών μεταβάσεων' στην οποία δεν απαιτείται η γνώση του αριθμού αυτού. Η ώρα εκκίνησης είναι πάντα 23:50 (διότι η προσομοίωση ξεκινάει στις 00:00), συνεπώς η μήτρα αυτή δεν απαιτείται να περιλαμβάνει την ώρα της ημέρας ως σχετικό παράγοντα.

3.8. Συσκευές (appliance)

Αντικείμενα που ανήκουν στην κλάση αυτή αποτελούν "συσκευές", δηλαδή αντικείμενα που έχουν τη δυνατότητα να καλύπτουν ανάγκες των κατοίκων του κτιρίου χρησιμοποιώντας

ορισμένη ηλεκτρική ισχύ για κάθε δεκάλεπτο λειτουργίας τους. Στα χαρακτηριστικά τους περιλαμβάνονται:

- Ο τύπος τους (πχ. ψυγείο, πλυντήριο)
- Η κατανάλωση ενέργειας που έχουν και ο κύκλος κατανάλωσης ενέργειας που έχουν αν η ζήτησή τους δεν είναι σταθερή
- Η πληροφορία αν λειτουργούν “αυτόματα” (όπως πχ. το ψυγείο) ή απαιτείται ενεργοποίηση και χρήση άμεσα από κατοίκους του κτιρίου (πχ. στην ηλεκτρική κουζίνα)
- Συγκεκριμένο προφίλ χρήσης (πότε χρησιμοποιούνται) ανάλογα με την ανάγκη που καλύπτουν, τον τύπο της ημέρας, τον αριθμό ενεργών κατοίκων και την ώρα της ημέρας
- Η κατάσταση της συσκευής της στιγμή που εξετάζεται (ζήτηση ενέργειας, σημείο κύκλου λειτουργίας στο οποίο βρίσκεται)

Σε αυτή την κλάση περιλαμβάνονται μέθοδοι σχετικά με τους κύκλους λειτουργίας της κάθε συσκευής. Έτσι, σε κάθε δεκάλεπτο και για κάθε συσκευή, μας ενδιαφέρει να βρούμε την επόμενη κατάσταση της συσκευής. Αν η συσκευή ήταν ανενεργή, χρησιμοποιείται η μέθοδος “handle_appliances_turned_off” για να αποφασίσουμε αν θα γίνει έναυση της συσκευής, ενώ σε περίπτωση που ήταν ήδη ενεργή ή μόλις έγινε έναυση, υπολογίζεται η ζήτηση ενέργειας στο επόμενο δεκάλεπτο και εξετάζουμε την πιθανότητα απενεργοποίησης της συσκευής με χρήση της μεθόδου “get_power_usage”

3.9. Ζήτηση ισχύος συσκευών (electricity)

Στόχος της κλάσης αυτής είναι η εύρεση της συνολικής ζήτησης ενέργειας των συσκευών του κτιρίου για κάθε δεκάλεπτο του χρονικού διαστήματος που εξετάζεται. Αυτό επιτυγχάνεται με τις μεθόδους “get_electricity_demand” και “electricity_data_updater” οι οποίες καλούν τη μέθοδο “get_daily_electricity_demand” για κάθε ημέρα και αποθηκεύουν τα δεδομένα που αυτή παράγει. Η μέθοδος “get_daily_electricity_demand” αναλαμβάνει να προσομοιώσει τη συμπεριφορά κάθε συσκευής του κτιρίου για κάθε δεκάλεπτο μιας ημέρας, και να αποθηκεύσει τα παραγόμενα δεδομένα ζήτησης για κάθε δεκάλεπτο και κάθε συσκευή ξεχωριστά.

3.10. Φωτιστικό Σώμα (luminaire)

Η κλάση αυτή αποτελεί την ισοδύναμη της κλάσης “συσκευή” για τα φωτιστικά του κτιρίου: Κάθε μέλος της είναι ένα φωτιστικό σώμα, το οποίο έχει ως χαρακτηριστικά:

- Την ισχύ του σώματος
- Το προφίλ χρήσης του σώματος
- Την κατάσταση του φωτιστικού τη στιγμή που εξετάζεται (ζήτηση ενέργειας και σημείο κύκλου λειτουργίας)

Σε ένα “φωτιστικό” (αντικείμενο της κλάσης αυτής) περιλαμβάνονται μέθοδοι που προσομοιώνουν τον κύκλο λειτουργίας του. Συγκεκριμένα η “get_power_usage” εκτελείται σε κάθε δεκάλεπτο και ελέγχει αν το φωτιστικό πρέπει να ενεργοποιηθεί (μέσω της “get_light_level_switch_on_chance”), αν πρέπει να απενεργοποιηθεί (όταν τελειώσει ο κύκλος χρήσης του) και βρίσκεται επίσης η ζήτηση ενέργειας του φωτιστικού. Σε περίπτωση έναυσης, η διάρκεια έναυσης δίνεται από τη μέθοδο “get_on_duration”.

3.11. Φωτισμός (lighting)

Έχοντας ήδη μια σειρά φωτιστικών για κάθε κτίριο, η κλάση αυτή περιέχει μεθόδους με σκοπό την εύρεση της ζήτησης ενέργειας από τα φωτιστικά του κτιρίου για το εξεταζόμενο διάστημα. Με τις μεθόδους “get_lighting_demand” και “lighting_data_updater” το εξεταζόμενο χρονικό διάστημα χωρίζεται σε ημέρες και αποθηκεύονται τα παραγόμενα δεδομένα, ανεξάρτητα για κάθε φωτιστικό. Η μέθοδος “get_daily_lighting_demand” εκτελεί την προσομοίωση για το χρονικό διάστημα μιας ημέρας, βρίσκοντας πρώτα για κάθε δεκάλεπτο τις συνθήκες εξωτερικού φωτισμού, συνυπολογίζοντας τα δεδομένα ενεργών κατοίκων και την κοινή χρήση φωτισμού για το δεκάλεπτο αυτό και στη συνέχεια, για κάθε φωτιστικό σώμα, προσομοιώνοντας τη συμπεριφορά του για το δεκάλεπτο που εξετάζεται.

3.12. Κοινή βιβλιοθήκη (shared)

Η βιβλιοθήκη αυτή δεν αποτελεί κλάση της εφαρμογής. Αντίθετα περιλαμβάνει μεθόδους στις οποίες όλες τις άλλες κλάσεις έχουν πρόσβαση για να επιτελέσουν διάφορες βιοηθητικές λειτουργίες. Τέτοιες μέθοδοι έχουν ρόλους όπως φόρτωση δεδομένων από το σκληρό δίσκο και αποθήκευση σε αυτόν από/πρός διάφορες μορφές, τη μετατροπή δομών τύπου DataFrame από ierarchikés σε μη ierarchikés, εύρεση του μήνα αν γνωρίζουμε μόνο τον αριθμό της ημέρας κ.α. Τέτοιες μέθοδοι, αν και απαραίτητες για τη λειτουργία πολλών τμημάτων της εφαρμογής, δεν ταίριαζαν σε κανένα από αυτά, συνεπώς η ύπαρξη αυτής της κοινής βιβλιοθήκης στην οποία θα ταιριάζουν βιοηθητικές συναρτήσεις κρίθηκε αναγκαία.

3.13. Απόδοση εφαρμογής

Κατά τη διάρκεια της ανάπτυξης της εφαρμογής, τηρήθηκαν όχι μόνο κανόνες ορθού προγραμματισμού με σκοπό την καλή ταχύτητα εκτέλεσης, αλλά έγινε ανάλυση της εφαρμογής και διάφορες βελτιστοποιήσεις με σκοπό την αποδοτικότερη εκτέλεση. Επιπλέον, η χρήση των βιβλιοθηκών numpy και pandas βοήθησε σημαντικά στην απόδοση της εφαρμογής διότι οι μέθοδοι και οι δομές οι οποίες περιλαμβάνονται σε αυτές τις βιβλιοθήκες είναι γραμμένες στη γλώσσα προγραμματισμού C της οποίας οι χρόνοι εκτέλεσης είναι γενικά καλύτεροι από της γλώσσας python. Τέλος, επιλέχθηκε η ανάλυση να είναι ανά δεκάλεπτο και όχι καλύτερη, για λόγους απόδοσης (πχ. ανάλυση ανά λεπτό αναμένεται να αυξήσει το χρόνο εκτέλεσης κατά μια τάξη μεγέθους). Κατά συνέπεια, η εφαρμογή έχει ικανοποιητικό χρόνο εκτέλεσης για τις ανάγκες τις προσομοίωσης. Μετά από δοκιμές που

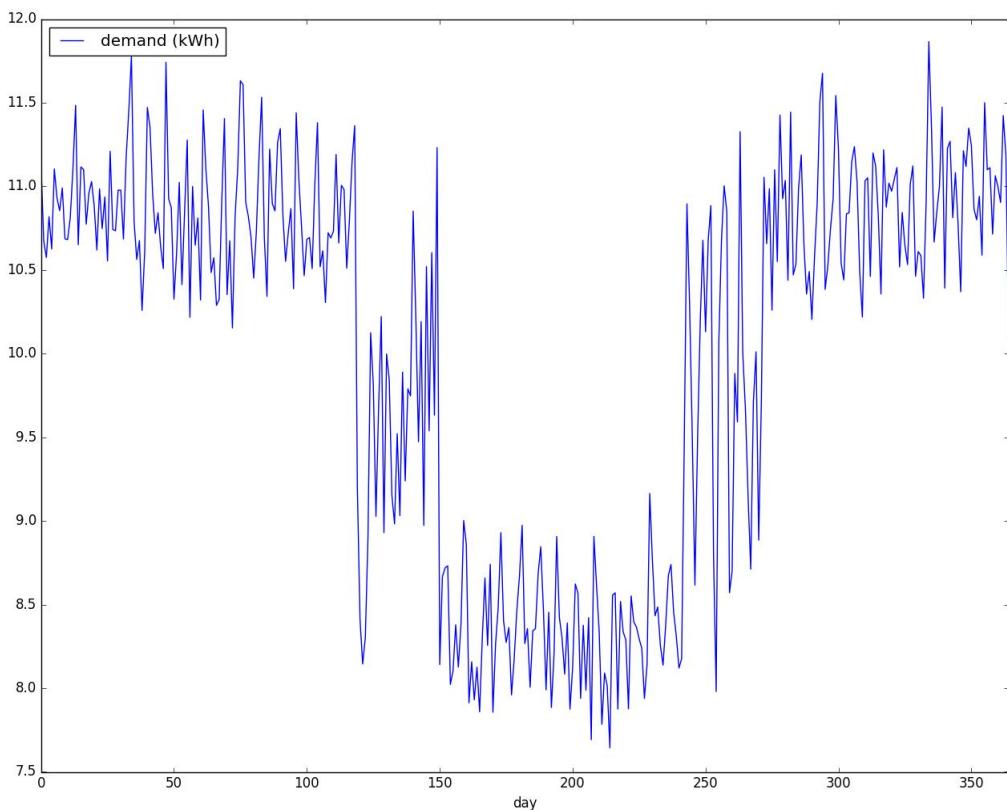
έγιναν σε ένα σύγχρονο επεξεργαστή (Intel i5 6600), και με χρήση ενός μόνο πυρήνα του επεξεργαστή (διότι η υλοποίηση δεν κρίθηκε σκόπιμο να υποστηρίζει ταυτόχρονη χρήση άνω του ενός πυρήνα), ο χρόνος προσομοίωσης 30 ημερών για 10 κτίρια είναι ~1 λεπτό. Όμοια για ένα κτίριο επί ένα έτος ο χρόνος εκτέλεσης είναι και πάλι ~1 λεπτό, ενώ αντίστοιχα για 1 έτος και 10 κτίρια ο χρόνος ανεβαίνει σε 10 λεπτά. Εφόσον η αύξηση είναι αναμενόμενα γραμμική, μπορεί να υποτεθεί με ασφάλεια πως η απόδοση θα παραμείνει ανάλογη με τυχόν αύξηση του χρόνου προσομοίωσης ή του αριθμού των κτιρίων, υπολογίζοντας περίπου ένα λεπτό για κάθε κτίριο και για κάθε έτος. Οι χρόνοι αυτοί είναι καθ' όλα αποδεκτοί για τις ανάγκες της προσομοίωσης, κάνοντας εφικτή την προσομοίωση 1000 ή περισσοτέρων κτιρίων για 365 ημέρες σε ένα κοινό προσωπικό υπολογιστή. Τέτοια δείγματα προσομοίωσης επαρκούν για την παραγωγή δεδομένων ικανοποιητικής ακρίβειας, συνεπώς περεταίρω βελτιστοποίηση του κώδικα με σκοπό τη βελτίωση του χρόνου εκτέλεσης δεν κρίνεται απαραίτητη.

4. Αποτελέσματα – Συζήτηση

4.1. Παρουσίαση αποτελεσμάτων

Ακολουθούν διαγράμματα από δεδομένα που παρήχθησαν κατά την εκτέλεση της εφαρμογής, για διάφορους τύπους κτιρίων και χρονικές στιγμές. Ορισμένα από αυτά έχουν αθροιστεί σε αναλύσεις διαφορετικές της ανάλυσης δεκαλέπτου για να εξαχθούν ευρύτερα συμπεράσματα. Τα κτίρια είναι διαφορετικά μεταξύ τους, συνεπώς διαθέτουν διαφορετικούς συνδυασμούς συσκευών κάθε φορά. Στα διαγράμματα όπου το δείγμα περιλαμβάνει πάνω από ένα κτίριο, τα δεδομένα αντιπροσωπεύουν ολόκληρο το δείγμα που αναφέρεται, και παρουσιάζονται οι μέσες τιμές των αποτελεσμάτων για τα κτίρια αυτά.

Σχ. 4.1. Ζήτηση κτιρίων ενός κατοίκου, δείγμα 100 κτιρίων, ανάλυση ημερήσια, διάρκεια ένα έτος. Ζήτηση σε kWh για κάθε ημέρα του έτους.



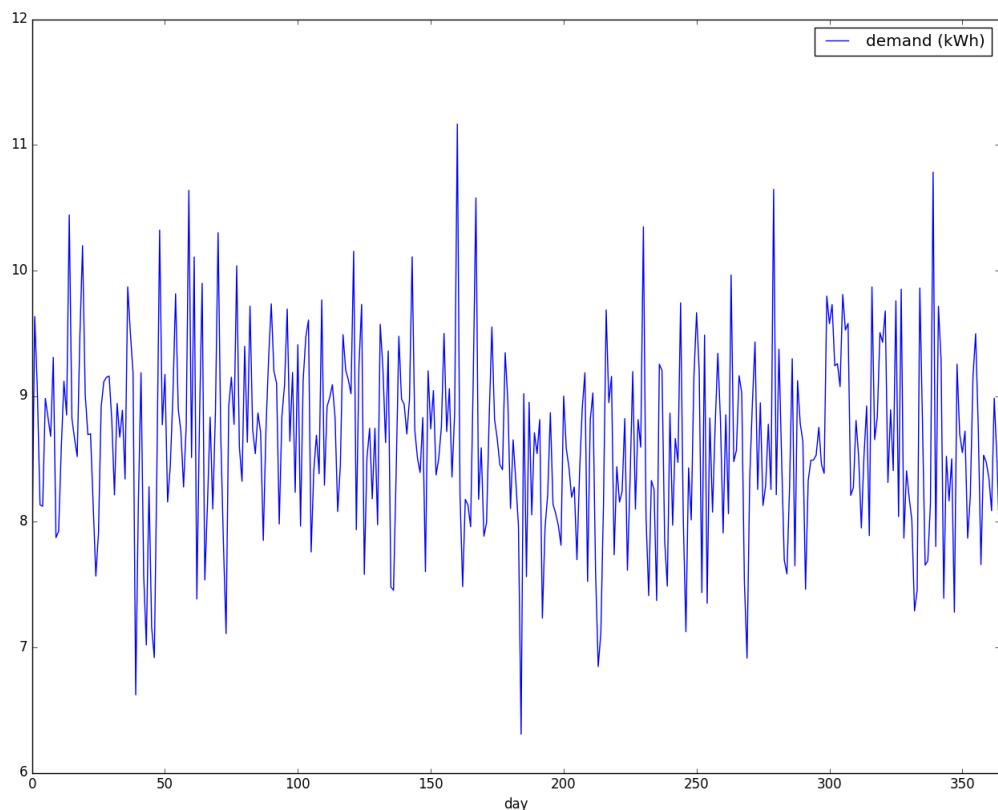
Στο ετήσιο δείγμα το πρώτο πράγμα που παρατηρείται είναι η σημαντική μείωση ζήτησης ενέργειας το καλοκαίρι. Αυτό συμβαίνει λόγω του γεγονότος πως μέρος των κτιρίων του δείγματος διαθέτει ηλεκτρική συσκευή θέρμανσης χώρου, συνεπώς το χειμώνα που αυτή είναι ενεργή η ζήτηση ενέργειας είναι αρκετά μεγαλύτερη από την αντίστοιχη το καλοκαίρι. Την ίδια εξήγηση έχει και η μεγάλη διακύμανση της ζήτησης κατά τη διάρκεια της άνοιξης και του φθινοπώρου. Εκείνες τις εποχές οι συσκευές θέρμανσης χώρου δε λειτουργούν συνεχώς, συνεπώς τις ημέρες που λειτουργούν η ζήτηση πλησιάζει τη ζήτηση

το χειμώνα, ενώ τις ημέρες που δε λειτουργούν, πλησιάζει τη ζήτηση το καλοκαίρι. Για δεδομένα που αντιπροσωπεύουν τον Ελληνικό χώρο, θα περίμενε κανείς να δει την ανάστροφη συμπεριφορά: Λόγω ύπαρξης πολλών κλιματιστικών συσκευών και ελάχιστων ηλεκτρικών συσκευών θέρμανσης (μιας και η θέρμανση στην Ελλάδα γίνεται σε μεγάλο βαθμό με λέβητα πετρελαίου/φυσικού αερίου - που δεν προσομοιώνεται στην εφαρμογή), η ζήτηση το καλοκαίρι ξεπερνάει την αντίστοιχη το χειμώνα. Αυτό δε φαίνεται στο διάγραμμα, διότι λόγω αδυναμίας εύρεσης κατάλληλων δεδομένων κατοχής συσκευών, χρησιμοποιήθηκαν δεδομένα που αντιπροσωπεύουν τη Μ. Βρετανία.

Ένα δεύτερο θέμα που αξίζει να αναφερθεί είναι πως η ζήτηση δε μηδενίζεται ποτέ. Αυτό, όπως θα φανεί και σε επόμενο διάγραμμα δε συμβαίνει σε μια κατοικία, στην οποία η ζήτηση μπορεί να μηδενιστεί για κάποια διαστήματα μέσα στη μέρα. Σε ένα δείγμα 100 κτιρίων όμως, λόγω ετεροχρονισμών, κάθε στιγμή υπάρχει κάποια ζήτηση, της οποίας ο μέσος όρος που φαίνεται στο διάγραμμα δε θα είναι ποτέ μηδέν.

Τέλος, η βραχυπρόθεσμη περιοδικότητα που φαίνεται στη ζήτηση κατά τη διάρκεια του έτους, έχει τις ρίζες τις τόσο στη διαφορετική συμπεριφορά που έχουν οι κάτοικοι μεταξύ καθημερινών και αργιών, όσο και στη στοχαστικότητα που έχει η ζήτηση σε οικιακά κτίρια.

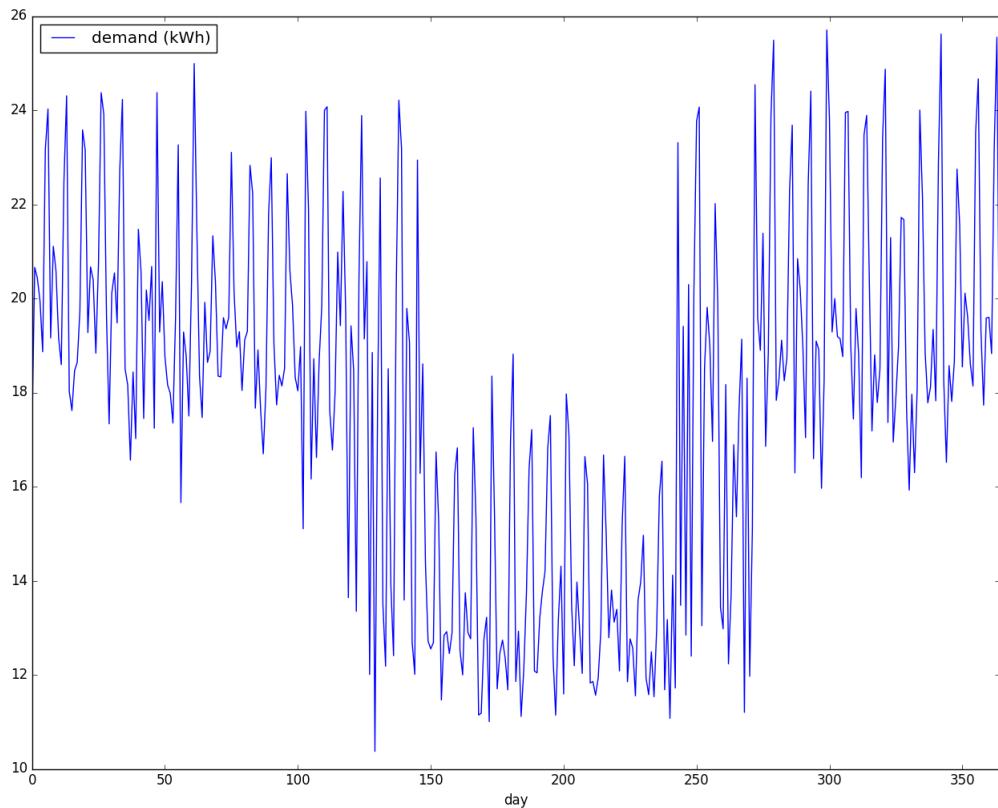
Σχ. 4.2. Ζήτηση κτιρίων ενός κατοίκου, δείγμα 10 κτιρίων, ανάλυση ημερήσια, διάρκεια ένα έτος. Ζήτηση σε kWh για κάθε ημέρα του έτους.



Σε αυτό το διάγραμμα παρατηρούμε πως η βραχυπρόθεσμη περιοδικότητα παραμένει, αλλά η εποχιακή περιοδικότητα εξαφανίστηκε. Αυτό συνέβη γιατί πιθανότατα το δείγμα

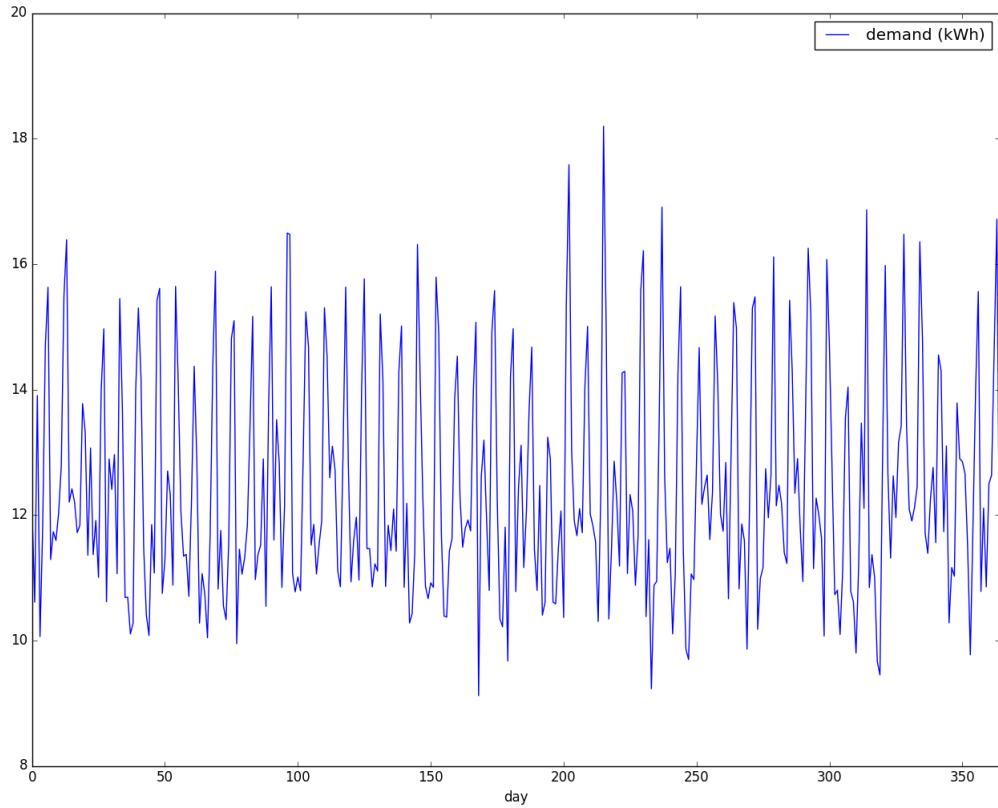
είναι πολύ μικρό, και έτυχε τα 10 κτίρια που δημιουργήθηκαν να μη χρησιμοποιούν ηλεκτρισμό για θέρμανση. Η υπόθεση αυτή ενισχύεται από την παρατήρηση πως η ζήτηση ενέργειας στο διάγραμμα έχει τιμές παρόμοιες με τη ζήτηση του Σχ.6.1. για το καλοκαίρι, όταν οι συσκευές θέρμανσης είναι απενεργοποιημένες.

Σχ. 4.3. Ζήτηση κτιρίων τριών κατοίκων, δείγμα 10 κτιρίων, ανάλυση ημερήσια, διάρκεια ένα έτος. Ζήτηση σε kWh για κάθε ημέρα του έτους.



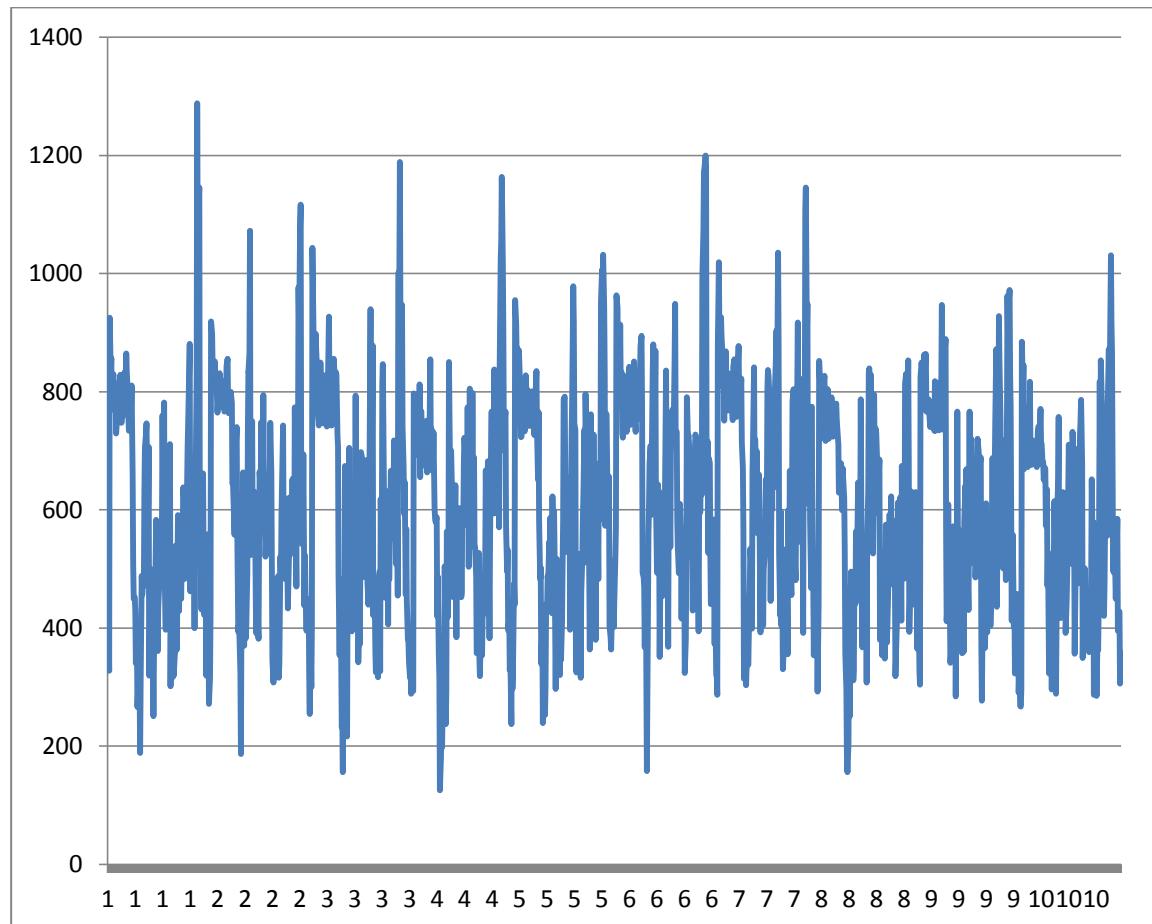
Στο διάγραμμα παρατηρούμε πως υπάρχει ξανά εποχιακότητα στη ζήτηση. Όπως αναφέρθηκε και παραπάνω, η ύπαρξη ή όχι αυτής έχει σχέση με το δείγμα, και δεν αποτελεί σημείο αναφοράς. Την ορθή λειτουργία του μοντέλου επιβεβαιώνει η διαφορά στη ζήτηση που έχει το κτίριο τριών κατοίκων από το κτίριο των δύο κατοίκων, η οποία είναι σαφώς μεγαλύτερη, της τάξης του 30%, γεγονός αναμενόμενο, καθώς η ζήτηση δεν αυξάνει γραμμικά με τον αριθμό των κατοίκων, λόγω κοινής χρήσης συσκευών και φωτιστικών.

Σχ. 4.4. Ζήτηση κτιρίων τυχαίου αριθμού κατοίκων, δείγμα 10 κτιρίων, ανάλυση ημερήσια, διάρκεια ένα έτος. Ζήτηση σε kWh για κάθε ημέρα του έτους.



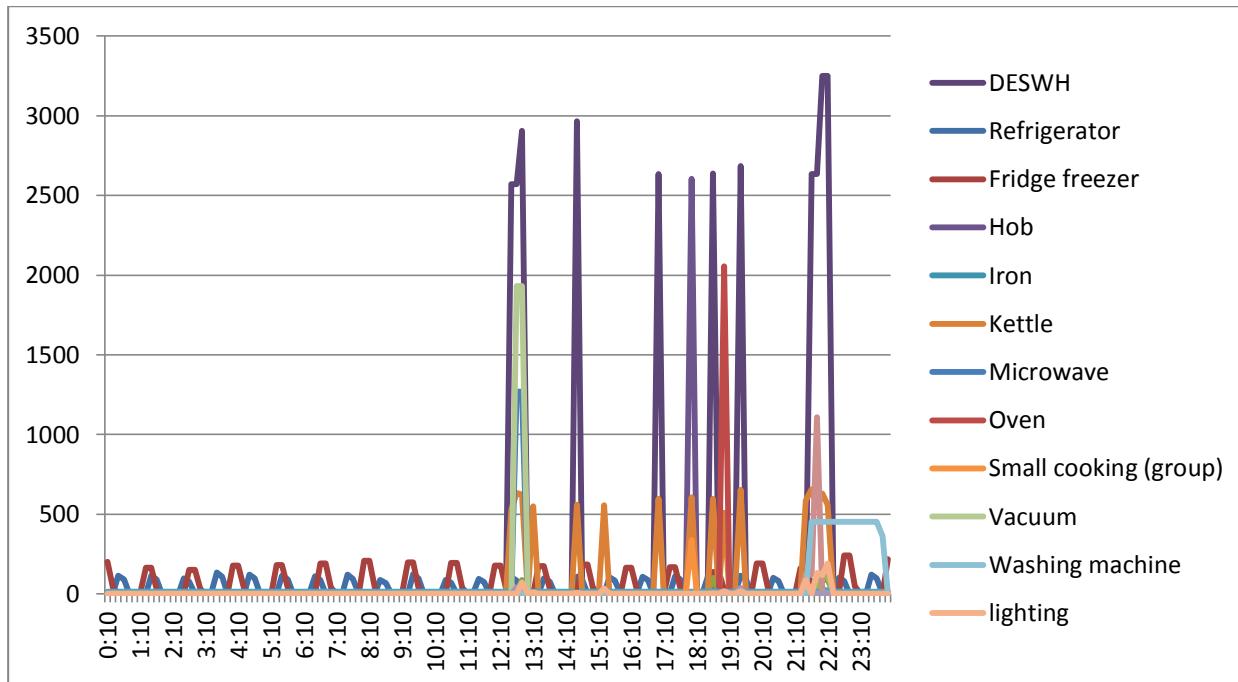
Σε δείγμα που αποτελείται από δέκα κτίρια που έχουν τυχαίο αριθμό κατοίκων (μπορεί να έχουν από έναν έως πέντε κατοίκους, με πιο πιθανά κτίρια με δύο ή τρείς κατοίκους) παρατηρούμε (αγνοώντας τη μη εποχιακότητα στη ζήτηση που έχει ήδη αναλυθεί εκτενώς) πως η ζήτηση έχει τιμές παρόμοιες με αυτές που έχει ένα κτίριο τριών κατοίκων.

Σχ. 4.5. Ζήτηση κτιρίων δύο κατοίκων, δείγμα 100 κτιρίων, δεδομένα ανάλυσης δεκαλέπτου για διάστημα δέκα ημερών. Στον κάθετο άξονα φαίνεται η ζήτηση ισχύος σε Watt, ενώ στον οριζόντιο η ημέρα (από 1η έως 10η)



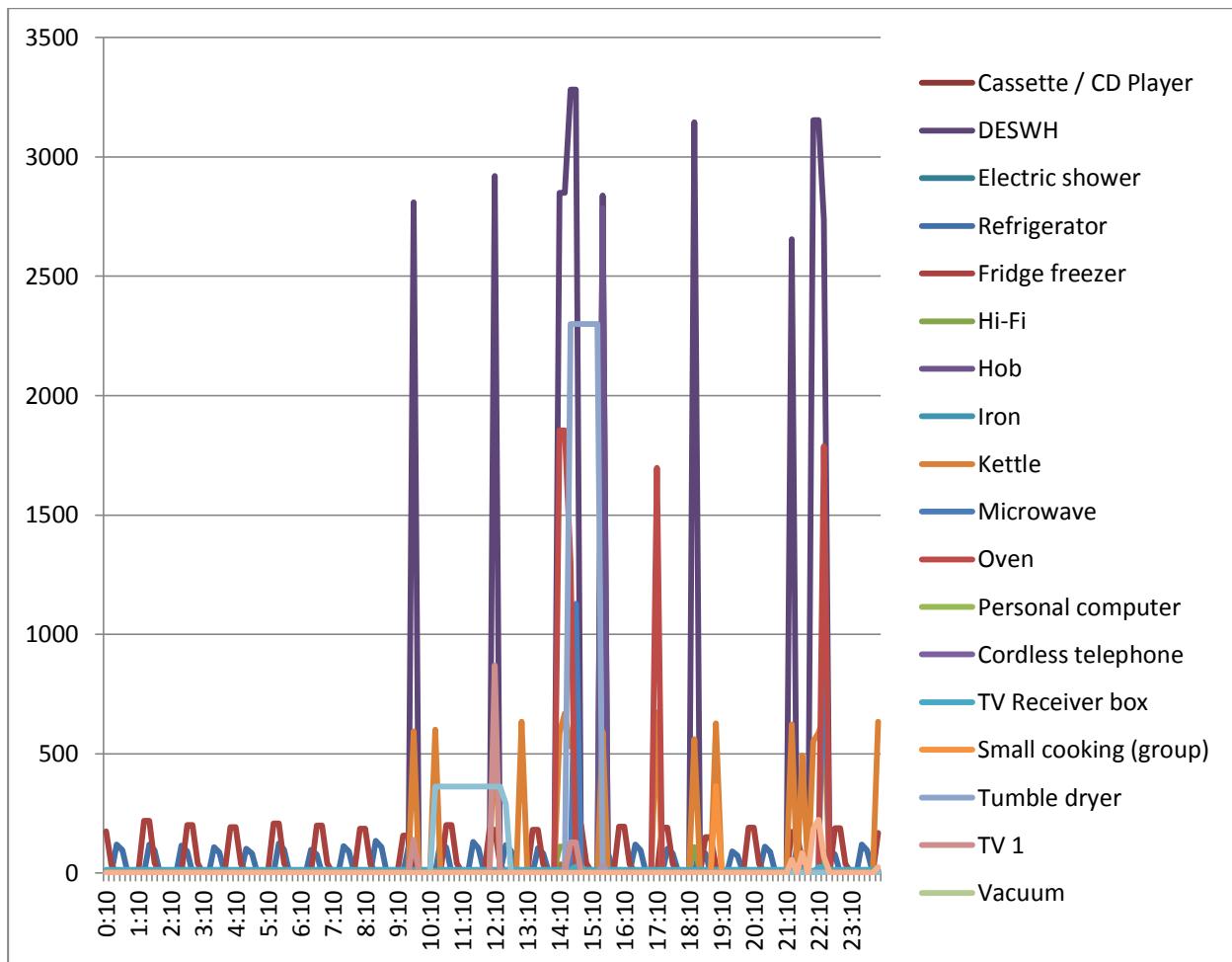
Στο διάγραμμα παρατηρείται καταρχήν μια καθημερινή διακύμανση στη ζήτηση, αναμενόμενη λόγω του κύκλου ημέρας/νύχτας τον οποίο και ακολουθούν οι δραστηριότητες των κατοίκων. Παρατηρούνται τιμές ζήτησης ισχύος από 200 έως 1200Watt, οι οποίες φαίνονται σωστές, αν λάβουμε υπόψη πως απεικονίζουν ζήτηση ισχύος στο δεκάλεπτο που εξετάζεται κάθε φορά. Για παράδειγμα, σταθερή ζήτηση 500Watt, θα οδηγούσε σε κατανάλωση 500Wh κάθε ώρα ή 12kWh κάθε ημέρα αριθμός ο οποίος είναι σαν τάξη μεγέθους κοντά στα δεδομένα που παρουσιάστηκαν παραπάνω.

Σχ. 4.6. Ζήτηση ενός κτιρίου δύο κατοίκων, δεδομένα ανάλυσης δεκαλέπτου για διάστημα μιας ημέρας, δεύτερη ημέρα του έτους. Ζήτηση σε kW.



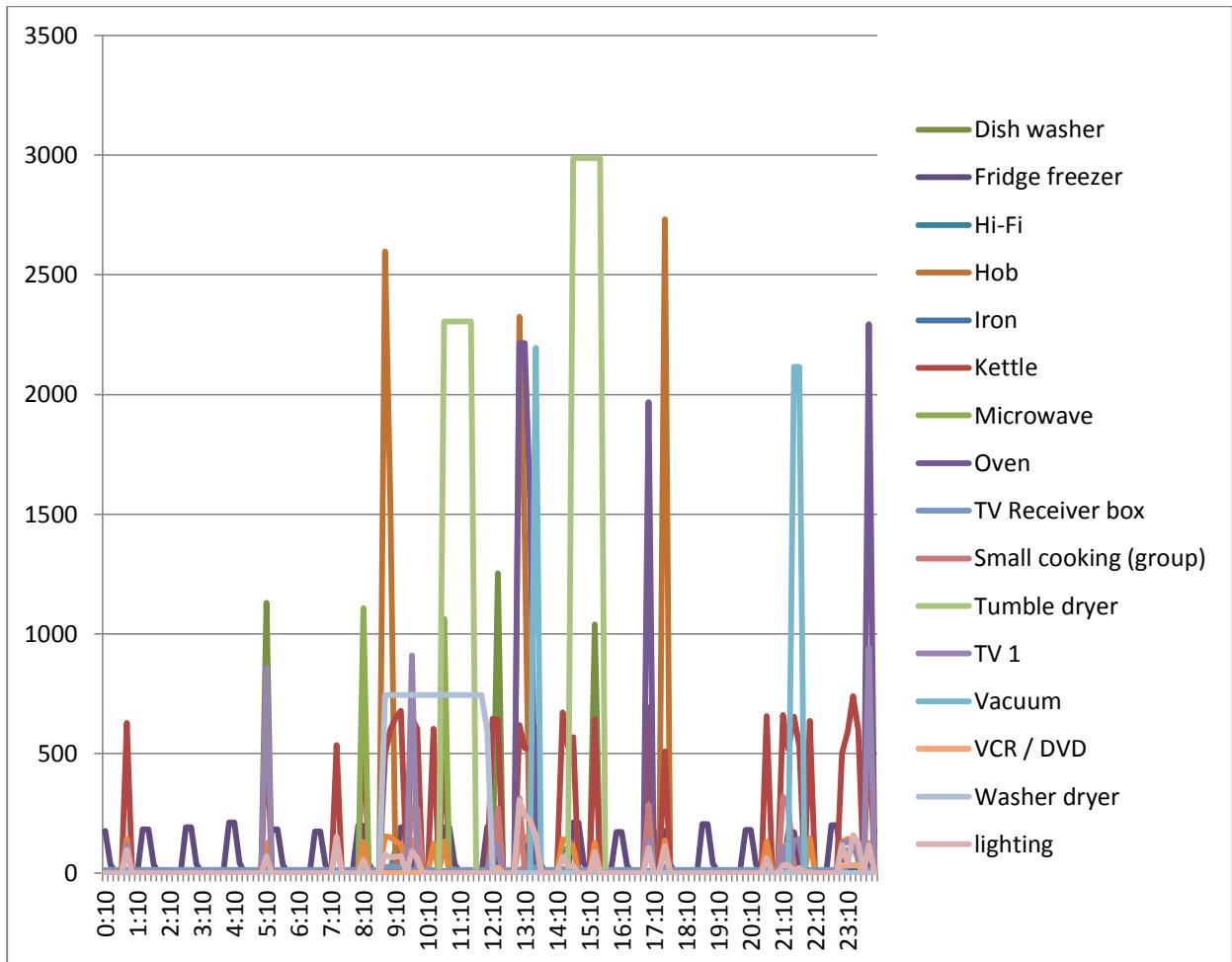
Στο διάγραμμα απεικονίζονται δεδομένα μιας ημέρας για κάθε συσκευή ξεχωριστά και για ένα μόνο τυχαίο κτίριο. Όπως φαίνεται, σημαντική ζήτηση υπάρχει από το μεσημέρι και μετά, πιθανότατα γιατί δεν υπάρχουν ενεργοί κάτοικοι στο κτίριο το πρωί. Από τις 12:00 και μετά βλέπουμε χρήση συσκευών έως και τις 24:00 όπου πιθανότατα οι κάτοικοι πάνε για ύπνο. Οι πιο ενεργοβόρες συσκευές, όπως αναμενόταν είναι οι: Ηλεκτρικός θερμαντήρας νερού (DESWH), φούρνος (Oven), ηλεκτρική σκούπα (Vacuum), μπρίκι (Kettle) και πλυντήριο (Washing machine). Επίσης παρατηρούμε ανοικτή τηλεόραση στις 22:00 και χρήση φωτιστικών το βράδυ. Τις ώρες που δεν υπάρχουν ενεργοί κάτοικοι, η ζήτηση εξαρτάται από συσκευές που λειτουργούν αυτόματα. Στο κτίριο αυτό αυτές είναι ο καταψύκτης και το ψυγείο. Η περιοδικότητα στη ζήτησή τους αιτιολογείται από την περιοδικότητα με την οποία αυτές οι συσκευές απαιτείται να λειτουργούν, καθώς δεν χρειάζεται να παράγουν ψύξη συνεχώς, παρά μόνο όταν η εσωτερική τους θερμοκρασία ξεπεράσει κάποιο όριο.

Σχ. 4.7. Ζήτηση ενός κτιρίου δύο κατοίκων, δεδομένα ανάλυσης δεκαλέπτου για διάστημα μιας ημέρας, 180η ημέρα του έτους. Ζήτηση σε kW.



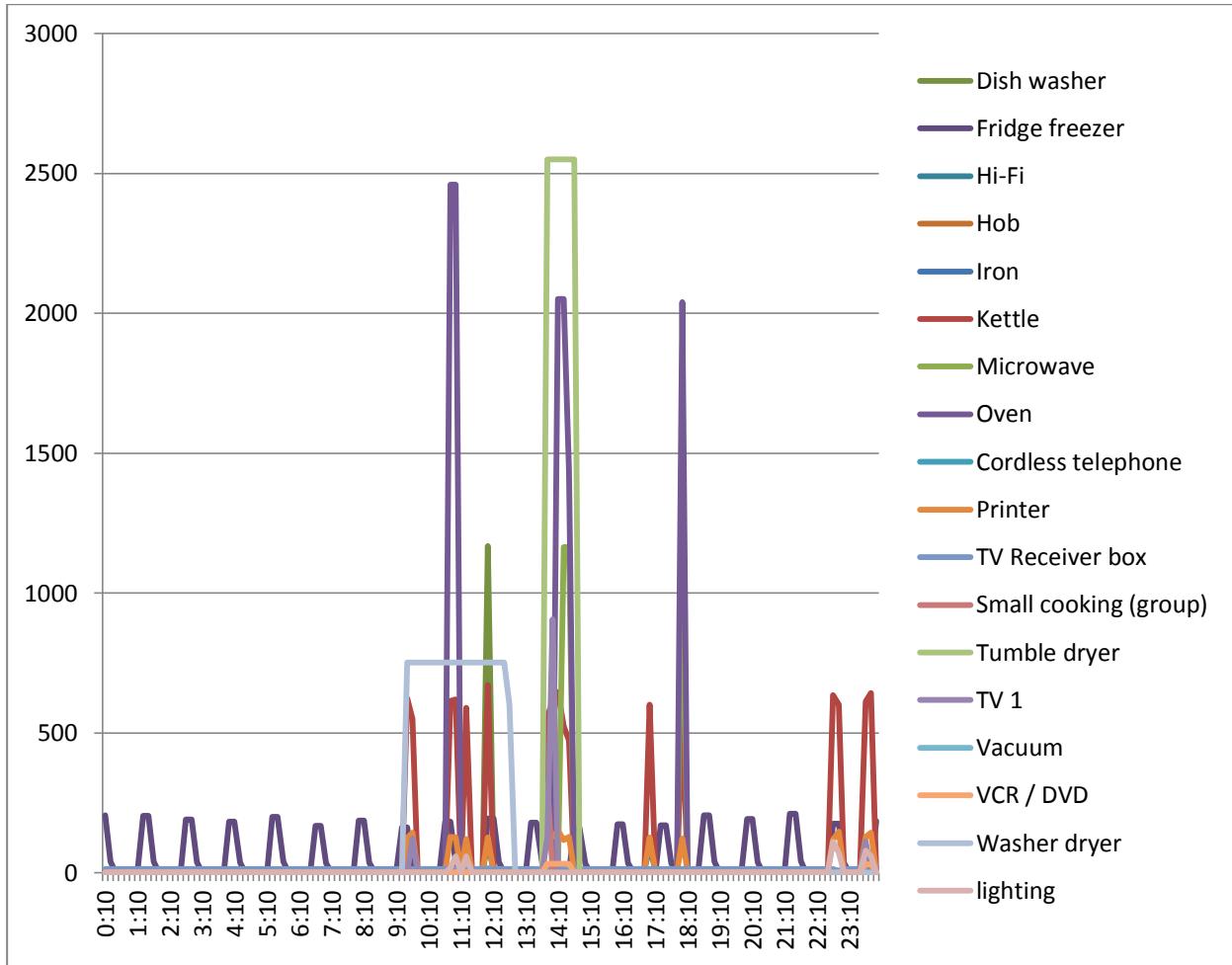
Την 180η ημέρα του έτους, για το ίδιο κτίριο παρατηρούμε ζήτηση παρόμοιας μορφής, με τη διαφορά ότι οι κάτοικοι ξεκινούν τις δραστηριότητές τους από τις 9:30. Επίσης βλέπουμε χρήση σεσουάρ στις 14:10, αν και τέτοιες λεπτομέρειες δεν είναι μεγάλης σημασίας, διότι για να εξαχθούν ασφαλή συμπεράσματα, απαιτείται να γίνει προσομοίωση σε ένα σύμπλεγμα κτιρίων και όχι σε ένα μεμονωμένο κτίριο.

Σχ. 4.8. Ζήτηση ενός κτιρίου τεσσάρων κατοίκων, δεδομένα ανάλυσης δεκαλέπτου για διάστημα μιας ημέρας, δεύτερη ημέρα του έτους. Ζήτηση σε kW.



Σε κτίριο τεσσάρων κατοίκων, η σημαντικότερη διαφορά που παρατηρείται είναι αρκετά μεγαλύτερος αριθμός δραστηριοτήτων σε σχέση με κτίρια δύο κατοίκων. Αυτό φυσικά είναι αναμενόμενη συμπεριφορά η οποία επιβεβαιώνει την ακρίβεια του μοντέλου και οδηγεί σε συνολικές καταναλώσεις γενικά μεγαλύτερες από αυτές σε κτίρια με δύο μόνο κατοίκους. Το κτίριο αυτό δε διαθέτει ηλεκτρικό θερμαντήρα νερού και παρατηρούμε ότι οι πιο ενεργοβόρες συσκευές είναι αυτές που έχουν σχέση με το μαγείρεμα, καθώς και η ηλεκτρική σκούπα και το στεγνωτήριο ρούχων (Tumble dryer)

Σχ. 4.9. Ζήτηση ενός κτιρίου τεσσάρων κατοίκων, δεδομένα ανάλυσης δεκαλέπτου για διάστημα μιας ημέρας, 180η ημέρα του έτους. Ζήτηση σε kW.



Την 180η ημέρα του έτους και για το ίδιο κτίριο των 4 κατοίκων, δε βλέπουμε αξιοσημείωτες διαφορές στη ζήτηση.

4.2. Συζήτηση

Από τα διαγράμματα είναι σαφές ότι τα δεδομένα ζήτησης ηλεκτρικής ενέργειας που παράγονται από την εφαρμογή είναι της αναμενόμενης μορφής. Καταρχήν, παρατηρούμε περιοδικότητα μέσα στην ημέρα, η οποία σχετίζεται με τις συνήθειες των κατοίκων. Η ζήτηση ενός μόνο κτιρίου εμφανίζει έντονες κορυφές, οι οποίες είναι επίσης αναμενόμενες, διότι οι συσκευές που καθορίζουν τη ζήτηση χρησιμοποιούνται μόνο λίγες ώρες κάθε ημέρα σε κάθε ανεξάρτητο κτίριο και όχι συνεχόμενα. Λόγω ετεροχρονισμών όμως, η ζήτηση ενός ολόκληρου συμπλέγματος από κτίρια, παρουσιάζει μεν ημερήσια περιοδικότητα αλλά όχι με τόσο έντονες κορυφές και χωρίς μηδενισμούς της ζήτησης.

Επιπλέον παρατηρούμε πως η ζήτηση μέσα σε ένα έτος είναι της αναμενόμενης τάξης μεγέθους, και μάλιστα πολύ κοντά σε πραγματικά στατιστικά δεδομένα, όπως τόσο τα δεδομένα για το 2015 που παρουσιάστηκαν από την στατιστική υπηρεσία της Μ. Βρετανίας [28] όσο και τα δεδομένα για το 2012 που παρουσιάστηκαν από την ΕΛΣΤΑΤ [29]. Και στις

δύο έρευνες αυτές, βλέπουμε πως η μέση ζήτηση ενέργειας για ένα νοικοκυριό μέσα σε ένα έτος πλησιάζει τις 4000kWh (Μ.Βρετανία: 3994kWh, Ελλάδα: 3750kWh). Αντίστοιχα, τα παραγόμενα αποτελέσματα (Σχ. 4.1, 4.2, 4.3, 4.4.) δείχνουν πως η ζήτηση που προβλέπει το μοντέλο είναι η εξής:

- Για νοικοκυριά 1 κατοίκου (Σχ. 4.1 & 4.2), κατά μέσο όρο περί τις 4000kWh/έτος αν συμπεριλάβουμε και συσκευές θέρμανσης, και περί τις 3100kWh/έτος αν όχι
- Για νοικοκυριά 3 κατοίκων (Σχ. 4.3), κατά μέσο όρο περί τις 5900kWh/έτος αν συμπεριλάβουμε και συσκευές θέρμανσης, και περί τις 4400kWh/έτος αν όχι
- Για νοικοκυριά τυχαίου αριθμού κατοίκων (Σχ. 4.4), κατά μέσο όρο περί τις 4400kWh/έτος χωρίς ηλεκτρικές συσκευές θέρμανσης

Τέλος, όπως αναλύθηκε στα Σχ. 4.1 και Σχ. 4.3, επειδή τα δεδομένα που χρησιμοποιήθηκαν είναι βασισμένα σε στατιστικές έρευνες που έγιναν στην Αγγλία, υπάρχει εποχιακότητα στη ζήτηση μη αντιπροσωπευτική για Ελλάδα, με αυξημένη ζήτηση το χειμώνα και όχι το καλοκαίρι. Αν τα δεδομένα ήταν για την Ελλάδα, θα αναμέναμε αυξημένη ζήτηση και χειμώνα και καλοκαίρι σε σχέση με το υπόλοιπο έτος, με μέγιστη ζήτηση το καλοκαίρι.

5. Συμπεράσματα

Η εφαρμογή σχεδιάστηκε με σκοπό την πρόβλεψη της ζήτησης ηλεκτρικής ισχύος και ενέργειας για κτίρια του οικιακού τομέα. Όπως φαίνεται από τα διαγράμματα, ο σκοπός αυτός επιτεύχθηκε και η εφαρμογή παράγει συνθετικά δεδομένα ζήτησης ηλεκτρικής ενέργειας σε κτίρια για τα ζητούμενα χρονικά διαστήματα και για συγκεκριμένο αριθμό και τύπο κτιρίων, τα οποία συγκλίνουν με πραγματικά. Επιπλέον, τα αποτελέσματα που παρουσιάστηκαν στο κεφάλαιο 4 αλλά και όπως αναφέρεται στα μοντέλα των Richardson I. et al. [5] [6] [7] στα οποία βασίστηκε η υλοποίηση, τα παραγόμενα δεδομένα του μοντέλου έχουν πολύ καλή συσχέτιση με την πραγματικότητα.

Οι βελτιώσεις που έγιναν στο αρχικό μοντέλο, περιλαμβάνουν τη θεώρηση πως υπάρχουν ανεξάρτητοι παράγοντες που επηρεάζουν τη συμπεριφορά των κατοίκων, μικρές βελτιστοποιήσεις σε ορισμένα σημεία οι οποίες μειώνουν τους απαραίτητους υπολογισμούς με αποτέλεσμα την αποδοτικότερη λειτουργία της εφαρμογής, αλλά και το ίδιο το γεγονός πως έγινε υλοποίηση με τη μορφή εφαρμογής ηλεκτρονικού υπολογιστή.

Από τις βελτιώσεις αυτές, προστιθέμενη αξία δίνει η θεώρηση πως υπάρχουν ανεξάρτητοι παράγοντες που επηρεάζουν τη συμπεριφορά των κατοίκων. Γενικά οι παράγοντες αυτοί είναι περίπλοκο να αναλυθούν, αλλά η διάσπασή τους που έγινε στην εργασία αυτή φαίνεται ορθή καθώς τα τελικά αποτελέσματα συγκλίνουν με πραγματικά. Επιπλέον, η θεώρηση αυτή δίνει τη δυνατότητα μοντελοποίησης πιθανών αλλαγών στη συμπεριφορά των κατοίκων με πιθανή χρήση σε μετέπειτα μελέτες.

Επιπλέον, το γεγονός πως η εφαρμογή είναι γραμμένη σε μια δημοφιλή, εύκολα επεκτάσιμη και εύκολη στη χρήση, γλώσσα προγραμματισμού (python) της δίνει ιδιαίτερη αξία, καθώς είναι απλή η ενσωμάτωσή της σε εφαρμογές μεγαλύτερης κλίμακας που απαιτούν δεδομένα ζήτησης ενέργειας υψηλής ανάλυσης, όπως για παράδειγμα προσομοιώσεις και αναλύσεις εφαρμογών διαχείρισης ζήτησης (DSM) ή μικροδικτύων. Την αξία της χρήσης της εφαρμογής με τέτοιο σκοπό ενισχύει το γεγονός πως τα δεδομένα ζήτησης που παράγονται έχουν υψηλή ανάλυση (δεκαλέπτου), πράγμα που είναι συχνά απαιτούμενο για τέτοιου είδους εφαρμογές. Επίσης, οι ιδιαίτερα ικανοποιητικοί χρόνοι προσομοίωσης αποτελούν ακόμα ένα πλεονέκτημα. Με αυτόν τον τρόπο διευρύνεται το πεδίο εφαρμογής της εργασίας καθώς και η μετέπειτα πιθανή χρησιμότητά της.

Πιθανή αδυναμία του μοντέλου είναι το γεγονός πως κάθε κατάσταση εξαρτάται από την προηγούμενή της, η οποία πρέπει να βρεθεί πριν περάσουμε στην εύρεση κάθε επόμενης κατάστασης. Ειδικά στη συγκεκριμένη περίπτωση το μοντέλο βασίζεται σε Μαρκοβιανές αλυσίδες των οποίων τα παραγόμενα δεδομένα χρησιμοποιούνται από επόμενες αλυσίδες, ενώ επιπλέον η κατάσταση μιας αλυσίδας απαιτείται να είναι γνωστή πριν βρεθεί το επόμενο σημείο της. Για αυτό το λόγο, πιθανώς προκαλούνται καθυστερήσεις στην εκτέλεση της εφαρμογής, διότι γίνεται αδύνατος ο υπολογισμός των αποτελεσμάτων ταυτόχρονα για κάθε αλυσίδα ή πίνακα. Εφόσον όμως δείχθηκε πως ο χρόνος εκτέλεσης είναι ικανοποιητικός, το θέμα δεν αποτελεί σημαντικό πρόβλημα για την εφαρμογή.

Ακόμη, περιορισμό της μεθόδου αποτελεί το γεγονός πως βασίζεται στην ύπαρξη αναλυτικών στατιστικών δεδομένων για την προσομοίωση. Ακριβή δεδομένα δεν υπήρχαν διαθέσιμα για τον ελληνικό χώρο, συνεπώς έγινε χρήση δεδομένων από τη Μ. Βρετανία και, σε μερικά σημεία εκτιμήσεις για τις τιμές τους. Από τα αποτελέσματα όμως είναι σαφές πως η

χρήση αυτών των δεδομένων δεν αποτελεί σημαντικό πρόβλημα καθώς έχουν την αναμενόμενη μορφή. Πιθανά προβλήματα στα αποτελέσματα θα μπορούσαν πιθανότατα να διορθωθούν με κατάλληλη ρύθμιση των υπαρχόντων συντελεστών “calibration”. Σε αυτό το σημείο πρέπει επίσης να αναφερθεί ότι αυτός ο περιορισμός αποτελεί ταυτόχρονα και πλεονέκτημα της μεθόδου, αφού δίνει τη δυνατότητα, εφόσον υπάρχουν τα κατάλληλα δεδομένα, να γίνει προσομοίωση κτιρίων τα οποία βρίσκονται σε διαφορετικές περιοχές του κόσμου και των οποίων οι κάτοικοι έχουν συνήθειες που διαφέρουν σημαντικά μεταξύ τους. Μοντέλα που δεν προσομοιώνουν τις συνήθειες αυτές θα μπορούσαν να οδηγήσουν σε σφάλματα σε τέτοιες περιπτώσεις, αλλά για την εφαρμογή του συγκεκριμένου μοντέλου αρκεί να βρεθούν αντιπροσωπευτικά στατιστικά δεδομένα για να παραχθούν αντιπροσωπευτικά αποτελέσματα.

6. Επόμενα βήματα

Το μοντέλο και η εφαρμογή επαρκούν ήδη για την πρόβλεψη της οικιακής ζήτησης σε μεγάλα χρονικά διαστήματα, αλλά υπάρχει μια σειρά από πιθανές βελτιώσεις αλλά και εφαρμογές οι οποίες έχουν ενδιαφέροντας κατευθύνσεις για μελλοντική έρευνα.

Σημαντική βελτίωση θα μπορούσε να επιφέρει η προσθήκη ενός μοντέλου ζήτησης θερμικής/ψυκτικής ενέργειας για κάθε κατοικία, δεδομένης της εξωτερικής θερμοκρασίας αλλά και της τεχνολογίας μόνωσης. Αυτή τη στιγμή ο παράγοντας αυτός μοντελοποιείται αφαιρετικά και εν' μέρει στην τρέχουσα υλοποίηση. Δεδομένου όμως του γεγονότος πως η απαιτούμενη ενέργεια για θέρμανση και ψύξη ενός κτιρίου αποτελεί βασικό τμήμα της συνολικής ζήτησης ενέργειας, η λεπτομερής προσομοίωσή του αναμένεται να επιφέρει αξιόλογες βελτιώσεις στη συνολική ακρίβεια του μοντέλου. Ως συνέπεια μιας τέτοιας βελτίωσης, το μοντέλο θα μπορούσε να προσομοιώσει και τη ζήτηση θερμικής/ψυκτικής ενέργειας, εκτός από την ηλεκτρική, και ακολούθως να υπολογίσει ζήτηση πρωτογενούς ενέργειας.

Επίσης, την ακρίβεια του μοντέλου θα μπορούσε να βελτιώσει η χρήση μετεωρολογικών προγνώσεων της επόμενης ημέρας ή εβδομάδας, με σκοπό την άντληση και χρήση δεδομένων όπως μέση θερμοκρασία ημέρας, ημερήσιο θερμοκρασιακό εύρος, ηλιακά θερμικά κέρδη, επίπεδα και χρόνος φωτισμού, καθώς τα δεδομένα αυτά έχουν έμμεση αλλά σημαντική επιρροή στη συνολική ζήτηση ενέργειας όλων των κτιρίων μιας περιοχής.

Σε περίπτωση που μετά από μελλοντικές προσθήκες ή για χρήση σε κάποια συγκεκριμένη εφαρμογή, η απόδοση της εφαρμογής πάψει να είναι ικανοποιητική, θα ήταν δυνατόν να επιτευχθούν πολύ μεγάλες βελτιώσεις στο χρόνο εκτέλεσης προσομοιώνοντας το κάθε κτίριο με μεθόδους παράλληλου προγραμματισμού. Εφόσον τα κτίρια είναι ξεχωριστά και ανεξάρτητα, ή υλοποίηση αυτής της τεχνικής δεν αναμένεται να είναι ιδιαίτερα περίπλοκη, ενώ ο χρόνος εκτέλεσης θα μπορούσε να βελτιωθεί ανάλογα με τους πυρήνες του επεξεργαστή που εκτελεί το πρόγραμμα (από τέσσερεις έως οκτώ φορές για κοινούς σημερινούς ηλεκτρονικούς υπολογιστές), εφόσον τα κτίρια που προσομοιώνονται είναι αρκετά. Επιπλέον, ορισμένα τμήματα του κώδικα τα οποία καθυστερούν την εκτέλεση της εφαρμογής, θα μπορούσαν να μεταφραστούν στη γλώσσα C και να καλούνται από τον υπόλοιπο κώδικα, πετυχαίνοντας μεγάλες βελτιώσεις στο χρόνο εκτέλεσης.

Πέραν των βελτιώσεων, ο λόγος για τον οποίο δημιουργήθηκε το μοντέλο, δεν είναι αυτή καθαυτή η πρόβλεψη της ζήτησης, αλλά η ενσωμάτωση του προβλεπτικού μοντέλου σε άλλα μοντέλα ή εφαρμογές που χρειάζονται τη ζήτηση ενέργειας ως δεδομένο. Προς αυτό το σκοπό, η ενσωμάτωση σε υπάρχουσες ή μελλοντικές εφαρμογές μεγάλης κλίμακας, είναι εφικτή διότι η υλοποίηση είναι σε μορφή εφαρμογής python στην οποία μπορούν να γίνουν κλήσεις με πολλούς διαφορετικούς τρόπους και αυτή να απαντήσει με τα απαιτούμενα δεδομένα κατά τις ανάγκες. Επίσης, επειδή οι παραγόμενες τιμές είναι υψηλής χρονικής ανάλυσης (ανά δέκα λεπτά), τα δεδομένα εξόδου είναι ιδιαίτερα συμβατά με αιολικές εφαρμογές, οι οποίες συνήθως έχουν ανάλυση ανά δεκάλεπτο.

Μια ιδανική τέτοια εφαρμογή είναι τα οικιακά μικροδίκτυα με δυνατότητα ανεξάρτητης από το δίκτυο λειτουργίας, στα οποία λόγω μικρού συνολικού αριθμού κατοικιών, και ελάχιστων προϋπαρχόντων στατιστικών δεδομένων ζήτησης, οι συμβατικοί τρόποι πρόβλεψης δεν επαρκούν. Η μελλοντική επέκταση του μοντέλου και της εφαρμογής θα μπορούσε να αποτελέσει τμήμα μιας ευρύτερης προσομοίωσης ενός μικροδικτύου και ενός μοντέλου πρόβλεψης-και-

εξισορρόπησης φορτίου [30], παρέχοντας έτσι τη δυνατότητα εκτίμησης των επιπτώσεων της ουσιαστικής ενσωμάτωσης των μικροδικτύων στην ενεργειακή αγορά και ανοίγοντας το δρόμο για την αύξηση της διεσπαρμένης παραγωγή ενέργειας σε δίκτυα μικρής κλίμακας. Αυτό αναμένεται να έχει σαν συνέπεια τόσο την ελάφρυνση των σημερινών αλλά και μελλοντικών υπερφορτωμένων δικτύων μεταφοράς ενέργειας, όσο και τη σημαντική βελτίωση της αποτελεσματικής λειτουργίας των μη διασυνδεδεμένων δικτύων όπως αυτά στο Ελληνικό νησιωτικό σύστημα.

Επιπλέον, η ενσωμάτωση σε εφαρμογές διαχείρισης της ζήτησης (DSM) είναι μια ενδιαφέρουσα προοπτική, καθώς η πρόβλεψη της οικιακής ζήτησης ισχύος αποτελεί αναπόσπαστο τμήμα της μοντελοποίησης, προσομοίωσης και σχεδιασμού τέτοιων εφαρμογών.

Ακόμη, ενδιαφέρον έχει και η μελέτη της επίδρασης αλλαγών στη συμπεριφορά των κατοίκων στη ζήτηση, πράγμα το οποίο μπορεί να μοντελοποιηθεί με ελάχιστες αλλαγές στην εφαρμογή.

Τέλος, η προσθήκη στο μοντέλο ενός agent-based τμήματος το οποίο θα μπορούσε να προσομοιώσει αλλαγές στη συμπεριφορά των κατοίκων ως ανταπόκριση σε εξωτερικούς παράγοντες όπως συγκεκριμένες κυβερνητικές πολιτικές, θα μπορούσε να χρησιμεύσει ιδιαίτερα στην αναγνώριση των επιπτώσεων των πολιτικών αυτών, αποτελώντας έτσι ένα χρήσιμο εργαλείο.

7. Βιβλιογραφία

- [1] N. Strachan and H. Dowlatabadi, “Distributed generation and distribution utilities” *Energy Policy*, issue 30, pp. 649-661, 2002.
- [2] A. Faruqui and S. George, “Quantifying Customer Response to Dynamic Pricing” *The Electricity Journal*, vol. 19, issue 4, pp. 53-63, 2005.
- [3] D. Kolokotsa, “The role of smart grids in the building sector” *Energy and Buildings*, issue 116, pp. 703-708, 2016.
- [4] H. Kirchoff, N. Kebir, K. Neumann and P. W. Heller, “Developing mutual success factors and their application to swarm electrification: microgrids with 100 % renewable energies in the Global South and Germany” *Journal of Cleaner Production*, issue 128, pp. 190-200, 2016.
- [5] I. Richardson, M. Thomson and D. Infield, “A high-resolution domestic building occupancy model for energy demand simulations” *Energy and Buildings*, issue 40, pp. 1560-1566, 2008.
- [6] I. Richardson, M. Thomson, D. Infield and C. Clifford, “Domestic electricity use: a high-resolution energy demand model” *Energy and Buildings*, issue 42, pp. 1878-1887, 2010.
- [7] I. Richardson, M. Thomson, D. Infield and A. Delahunty, “Domestic lighting: a high-resolution energy demand model” *Energy and Buildings*, issue 41, pp. 781-789, 2009.
- [8] F. Tanrisever, K. Derinkuyu and M. Heeren, “Forecasting electricity infeed for distribution system networks: An analysis of the Dutch case” *Energy*, issue 58, pp. 247-257, 2013.
- [9] G. Aneiros, J. Vilar and P. Raña, “Short-term forecast of daily curves of electricity demand and price” *Electrical Power and Energy Systems*, issue 80, pp. 96-108, 2016.
- [10] A. Capasso, W. Grattieri, R. Lamedica and A. Prudenzi, “A bottom-up approach to residential load modelling” *IEEE Transactions on Power Systems*, issue 9, pp. 957-964, 1994.
- [11] J. V. Paatero and P. Lund, “A model for generating household electricity load profiles” *International Journal of Energy Research*, issue 30, pp. 273-290, 2006.
- [12] M. Stokes, M. Rylatt and K. Lomas, “A simple model of domestic lighting demand” *Energy and Buildings*, issue 36, pp. 103-116, 2004.
- [13] J. Widén and E. Wäckelgård, “A high-resolution stochastic model of domestic activity patterns and electricity demand” *Applied Energy*, issue 87, pp. 1880-1892, 2010.
- [14] U. Wilke, F. Haldi, J.-L. Scartezzini and D. Robinson, “A bottom-up stochastic model to predict building occupants’ time-dependent activities” *Building and Environment*, issue 60, pp. 254-264, 2013.
- [15] R. Yao and K. Steemers, “A method of formulating energy load profile for domestic buildings in the UK” *Energy and Buildings*, issue 37, pp. 663-671, 2005.
- [16] D. A. Levin, Y. Peres and E. L. Wilmer, *Markov Chains and Mixing Times*, American Mathematical Society, 2008.
- [17] C. M. Grinstead and J. L. Snell, *Introduction to Probability*, American Mathematical Society, 2006.
- [18] “Wikipedia.org - Markov chain Monte Carlo”. Available: https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo.
- [19] L. G. Swan and I. V. Ugursal, “Modeling of end-use energy consumption in the residential sector: A review of modeling techniques” *Renewable and Sustainable Energy Reviews*, issue 13, pp. 1819-1835, 2009.
- [20] Ipsos-RSL and Office for National Statistics, *United Kingdom Time Use Survey*, 2000.
- [21] Python Software Foundation, “Python Programming Language”. Available: www.python.org.
- [22] B. Stroustrup, *The C++ Programming Language*, Pearson Education, Inc., 2013.
- [23] “Wikipedia.org - Modular Programming”. Available: https://en.wikipedia.org/wiki/Modular_programming.
- [24] M. Lutz, *Learning Python*, 2007.
- [25] “Wikipedia.org - Object-oriented_programming”. Available: https://en.wikipedia.org/wiki/Object-oriented_programming.
- [26] S. van der Walt, S. C. Colbert and G. Varoquaux, “The NumPy Array: A Structure for Efficient Numerical Computation” *Computing in Science & Engineering*, pp. 13, 22-30, 2011.
- [27] W. McKinney, “Data Structures for Statistical Computing in Python” σε *Proceedings of the 9th Python in Science Conference*, 2010.
- [28] UK National Statistics, Department of Business, Energy & Industrial Strategy, “Energy Consumption in the UK (2016) “ 2016. Available: <https://www.gov.uk/government/statistics/energy-consumption-in-the-uk>.
- [29] Ελληνική Στατιστική Αρχή, “Ερευνα Κατανάλωσης Ενέργειας στα Νοικοκυριά, 2011 - 2012” 2013. Available:

<http://www.statistics.gr/documents/20181/e74d6134-8c02-404e-a02b-aa6d959219e3>.

- [30] Y. Mengmeng and H. H. Seung, “Supply–demand balancing for power management in smart grid: A Stackelberg game approach” *Applied Energy*, issue 164, pp. 702-710, 2016.

8. Παράρτημα

8.1. Κώδικας προγράμματος

8.1.1. Main

```
import house
import occupancy
import occupancy_data
import electricity
import appliance_data
import lighting
import luminaire_data
import shared
import numpy

# This is the main module
# It contains all the paths for the data used in the model (used as optional args at init)
# It loads all the classes needed to generate a house. These are shared by all houses (exception: appliances are not)
# It can create houses as needed, which are then used to create occupancy and power drain scenarios and get the respective data

class Main:
    """The Main class"""

    def __init__(self, regenerate_trasitional_data = False,
                 occupancy_datapaths = {"base_datapath": "occupancy_data/base_data.xlsx", "initial_matrix": \
                                         "occupancy_data/initial_occupancy_transitions.csv", "general_matrix": \
                                         "occupancy_data/general_transitional_matrix.csv",
                                         "excel_multiindex": {"initial_occupancy": [0,1]}},
                 appliance_datapaths = {"base_appliances_data_path": \
                                         ("appliances", "electricity_data/appliances_base.xlsx"), \
                                         "appliances_data_path": ("electricity_data/appliances.csv"), \
                                         "activity_profiles_path": "electricity_data/activity_profiles.csv", \
                                         "relative_monthly_temperatures_path": \
                                         "electricity_data/relative_monthly_temperatures.csv", \
                                         "constants_datapath": "electricity_data/constants.csv"}, \
                 lighting_datapaths = {"irradiance_datapath": "lighting_data/irradiance_data.csv", \
                                         "luminaires_datapath": "lighting_data/house_types.csv", \
                                         "effective_occupancy_datapath": "lighting_data/effective_occupancy.csv", \
                                         "constants_datapath": "lighting_data/constants.csv", \
                                         "durations_datapath": "lighting_data/durations.csv"}, \
                 save_generated_data_folder = "generated_data"):

        # occupancy data gets regenerated only if we pass the regenerate_trasitional_data arg as true
        if regenerate_trasitional_data is True:
            self.occupancy_data_obj = \
                occupancy_data.OccupancyData(occupancy_datapaths["base_datapath"], \
                                              occupancy_datapaths["general_matrix"], occupancy_datapaths["excel_multiindex"])
            self.occupancy_data_obj.regen_general_transitional_matrix()
        # load occupancy data
```

```

    self.initial_occupancy_matrix = =
shared.load_dataframe_from_csv(occupancy_datapaths["initial_matrix"]) # base transitional data
    self.transitional_matrix = shared.load_dataframe_from_csv(occupancy_datapaths["general_matrix"])
# general transitional data
    # load electricity data
    self.electricity_obj = electricity.Electricity()
    self.occupancy_obj = occupancy.Occupancy(self.initial_occupancy_matrix, self.transitional_matrix)
    # load appliance data
    self.appliance_data_obj =
appliance_data.ApplianceData(appliance_datapaths["base_appliances_data_path"], \
                               appliance_datapaths["appliances_data_path"], \
                               appliance_datapaths["activity_profiles_path"], \
                               appliance_datapaths["relative_monthly_temperatures_path"], \
                               appliance_datapaths["constants_datapath"])
    # load lighting data
    self.lighting_obj = lighting.Lighting(lighting_datapaths["irradiance_datapath"], \
                                         lighting_datapaths["effective_occupancy_datapath"])
    # load luminaire data
    self.luminaire_data_obj = luminaire_data.LuminaireData(lighting_datapaths["luminaires_datapath"], \
                                         lighting_datapaths["constants_datapath"], lighting_datapaths["durations_datapath"])
    # remember the datapath to save the generated data
    self.save_generated_data_folder = save_generated_data_folder

def create_house(self, residents, name=""):
    return house.House(residents, self.occupancy_obj, self.electricity_obj, self.appliance_data_obj, \
                       self.lighting_obj, self.luminaire_data_obj, name)

# create and return as many house objects as specified,
# taking as input a 'residents' list containing the number of residents for each house
# if this list is not given, just assign a random residents number ranging from 1 to 5
def create_houses(self, number_of_houses, residents_list):
    houses = []
    for house_id in range(number_of_houses):
        # if residents is a list, use the id to find the number of residents, else use a random integer from 1-5
        # following a normal distribution with centrepoint=2.5 and deviation=1
        if residents_list:
            residents = residents_list[house_id]
        else:
            residents = round(numpy.random.normal(2.5,1))
        # bound the residents number
        if residents > 5: residents = 5
        elif residents < 1: residents = 1
        houses.append(self.create_house(residents, name="house_"+str(house_id)))
    return houses

# run the model for the specified house number and period, returning & saving data as needed
def run_model(self, houses_number, starting_weekday, duration, residents_list=False, \
             merge_and_save_data_daily_averages=False, save_one_house_data=False, \
             merge_and_save_data_10_min_averages=False):
    houses = self.create_houses(houses_number, residents_list)
    total_power_demand=[]
    for house in houses:

```

```

total_power_demand.append(house.get_power_demand(starting_weekday, duration))

if save_one_house_data:
    # save the days specified
    for day in save_one_house_data:
        day_power = total_power_demand[0][day]
        day_power.to_csv(self.save_generated_data_folder+'one_house_data_day_'+str(day)+'.csv')

if merge_and_save_data_daily_averages:
    merged_power_demand = shared.house_data_merger_into_daily_averages(total_power_demand)
    merged_power_demand.to_csv(self.save_generated_data_folder+'merged_daily_data.csv',
                               index_label=None)

if merge_and_save_data_10_min_averages:
    merged_power_demand = shared.house_data_merger_into_10_min_averages(total_power_demand)
    merged_power_demand.to_csv(self.save_generated_data_folder+'merged_10_min_data.csv',
                               index_label=None)

```

8.1.2. House

```

import numpy
import pandas

# This module implements the House class
# A house has appliances, luminaires and functions to determine occupancy and power demand for a
specific duration

class House:
    """This is a house"""

    def __init__(self, residents, occupancy, electricity, appliances_data, lighting, luminaire_data, name=""):
        self.residents = residents # this more than anything else characterizes the house
        self.name = name # give it a name
        # assign an instance of occupancy, electricity and lighting for power calculations
        self.occupancy = occupancy
        self.electricity = electricity
        self.lighting = lighting
        # assign appliances to the house at initialization
        self.appliances_data = appliances_data.assign_new_appliances()
        # also assign luminaires
        self.luminaires = luminaire_data.assign_new_luminaires()
        # get an irradiance threshold for the house
        self.irradiance_threshold = luminaire_data.get_house_irradiance_threshold()

    def get_occupancy(self, starting_weekday, duration):
        return self.occupancy.occupancy_generator(self.residents, starting_weekday, duration)

    def get_appliance_demand(self, starting_weekday, duration, occupancy):
        appliance_power_demand = self.electricity.get_electricity_demand(self.appliances_data, occupancy,
                           starting_weekday, duration)
        return appliance_power_demand

```

```

def get_luminaire_demand(self, starting_weekday, duration, occupancy):
    luminaire_power_demand = self.lighting.get_lighting_demand(self.luminaires, occupancy,
    starting_weekday, duration, self.irradiance_threshold)
    return luminaire_power_demand

# this will be the main power demand function
def get_power_demand(self, starting_weekday, duration):
    occupancy = self.get_occupancy(starting_weekday, duration)
    appliance_power_demand = self.get_appliance_demand(starting_weekday, duration, occupancy)
    luminaire_power_demand = self.get_luminaire_demand(starting_weekday, duration, occupancy)
    # concatenate the two demands in one table
    total_power_demand = pandas.concat([appliance_power_demand, luminaire_power_demand])
    # sum the y axis, to get the real power demand and append it to the dataframe as a bottom row
    sum_y_axis = total_power_demand.sum(axis=0, numeric_only=True).div(1000)
    sum_y_axis.name = 'Power sums in kW'
    total_power_demand = total_power_demand.append(sum_y_axis)
    # sum the x axis, divide it by 6 to get the real drain per hour and append to the dataframe as the final
    column
    total_power_demand['Energy sums in kWh'] = total_power_demand.sum(axis=1,
    numeric_only=True).div(6000)
    # fix the last cell
    total_power_demand.iat[-1,-1] = total_power_demand.iat[-1,-1]*1000
    return total_power_demand

```

8.1.3. Occupancy_data

```

import numpy
import pandas
import shared
import itertools

# This module, containing the TransitionalData class, is meant to handle all data manipulation relative to
occupancy.
# It creates a complete transitions multiindex dataframe by multiplying arrays containing possibilities for
occupancy to change
# depending on any possible factor. The result is a dataframe containing the cartesian product of all given
arrays

class OccupancyData:
    """Loads/saves/manipulates/regenerates transitional data"""

    def __init__(self, base_data_path, general_matrix_data_path, excel_multiindex):
        self.base_data_path = base_data_path
        self.general_matrix_data_path = general_matrix_data_path
        # this is a dictionary, with index name the data sheet name and value the multiindex rows
        # this is used to get an existing mutiindex when loading the base data. if the sheet name is not here, no
        multiindex is assumed
        self.excel_multiindex = excel_multiindex

        # regens and saves a new_general_transitional_matrix
        def regen_general_transitional_matrix(self):

```

```

new_general_transitional_matrix
self.create_general_transitional_matrix(self.load_base_dataframes())
shared.save_dataframe_to_csv(new_general_transitional_matrix,      self.general_matrix_data_path,
new_general_transitional_matrix.index.name)

# used to make an array to get used in generating the general_transitional_matrix
# is dynamic so it will work with any number of dataframes inside the dataframes list
def create_general_transitional_matrix_array(self, dataframes):
    # this function multiplies all rows of dataframe1 with all rows of dataframe2 and returns a 2d array of
    all products
        # has to be done as a separate function so that we can call it for every dataframe
    def multiply_all_rows(dataframe1, dataframe2):
        #whys is index line loaded as values?????
        tiled_list = numpy.tile(dataframe1, (len(dataframe2),1,1)).swapaxes (0,1) # tile the 1st array by the
length of the 2nd
        product = (dataframe2.as_matrix())*tiled_list # multiply the tiled first array with the 2nd to
produce the product
        return (list(itertools.chain(*product))) # return a flattened 2d array rather than a useless 3d
        # now just call the above function for each dataframe in dataframes (barring the first)
        new_transition_matrix_array = dataframes[0].values # we need to start with the first dataframe as
new_transition_matrix
        for index in range (len(dataframes)-1): # needs to run n-1 times
            new_transition_matrix_array      =      multiply_all_rows(new_transition_matrix_array,
dataframes[index+1])
        return new_transition_matrix_array

# this creates a multiindex for use with the general_transitional_matrix_array to create a hierachical
dataframe
def create_general_transitional_matrix_index(self, dataframes):
    iterables=[]
    names=[]
    for dframe in dataframes:
        # if it is already multiindex, separate the indices and append them to iterables
        if isinstance(dframe.index, pandas.core.index.MultiIndex):
            for index in dframe.index.levels:
                names.append(index.name)
                iterables.append(index.tolist())
        # if it is single index, create the index for the multiindex and append it to iterables
        else:
            names.append(dframe.index.name)
            iterables.append(dframe.index.tolist())
    new_multiindex = pandas.MultiIndex.from_product(iterables, names = names)
    # this is not efficient, but it will run only rarely
    iterable=[]
    for index_val in new_multiindex.values.tolist():
        iterable.append('(' + ','.join(str(x) for x in index_val) + ')')
    new_index = pandas.Index(iterable, name = '('+','.join(names)+')')
    return new_index

# creates the hierachical dataframe general_transitional_matrix as a multiindex pandas dataframe
def create_general_transitional_matrix(self, dataframes):
    new_transition_matrix = self.create_general_transitional_matrix_array(dataframes)

```

```

new_index = self.create_general_transitional_matrix_index(dataframes)
new_columns = dataframes[0].columns.values.tolist() # these are static and should be same between
all base matrices
    new_general_transitional_matrix = pandas.DataFrame(new_transition_matrix, index=new_index,
columns=new_columns) # make the dframe!
        # we still need to normalize the new_general_transitional_matrix before returning it,
        # since the possibilities do not sum to 1 as it is. to do it we just divide each value with its row's sum
        new_general_transitional_matrix = new_general_transitional_matrix.div(new_general_transitional_matrix.sum(axis=1), axis=0)
    return new_general_transitional_matrix

# decides if the sheet we need to load is multiindex or not, based on the self.xlsx_multiindex dictionary
# then correctly returns the dataframe
def determine_multiindex(self, sheet_name, basedata):
    if sheet_name in self.xlsx_multiindex:
        return basedata.parse(sheet_name, header=9, index_col=self.xlsx_multiindex[sheet_name]) # multiindex sheet
    else:
        return basedata.parse(sheet_name, header=9, index_col=0) # single index. assumes index is the first column

# loads the base data from the datafile
# if index_to_return=-1 it returns all dataframes, else it will return just the one with the index
def load_base_dataframes(self, sheet_to_return=-1):
    basedata = pandas.ExcelFile(self.base_data_path)
    dataframes = []
    if sheet_to_return == -1:
        for sheet_name in basedata.sheet_names:
            dataframes.append(self.determine_multiindex(sheet_name, basedata))
    return dataframes
    else:
        return self.determine_multiindex(sheet_to_return, basedata)

```

8.1.4. Appliance_data

```

import numpy
import pandas
import appliance as appliance_object # rename since the word appliance is already used quite a lot inside
this file
import shared

# This module implements the ApplianceData class
# It is mainly used to generate new appliances data based on the base_appliances.xlsx file,
# while taking into account the calibration and mean_active_occupancy variables.
# In addition, it is used to assign appliances to a house, returned as a tuple: (assigned_appliances,
assigned_appliances_names)

class ApplianceData:
    """Methods for loading/completing/regenerating the main csv file. Also Methods that assign appliances
to a house"""

    def __init__(self, base_appliances_data_path, complete_appliances_data_path, activity_profiles_path, \

```

```

relative_monthly_temperatures_path, constants_datapath):
    self.base_appliances_data_path = base_appliances_data_path
    self.complete_appliances_data_path = complete_appliances_data_path
    self.activity_profiles = shared.load_dataframe_from_csv(activity_profiles_path, [0,1])
    self.relative_monthly_temperatures = shared.load_dataframe_from_csv(relative_monthly_temperatures_path)
    self.constants = shared.load_dataframe_from_csv(constants_datapath, 0, None)

# randomly assigns appliances to the house (as a list of appliance objects)
def assign_new_appliances(self):
    appliances_data = shared.load_dataframe_from_csv(self.complete_appliances_data_path)
    assigned_appliances = []
    # for every possible appliance
    for device_type in appliances_data.index.tolist():
        # check if the house gets the appliance
        if numpy.random.random() < appliances_data.loc[device_type]['possibility_to_get']:
            # get the correct activity profile for the appliance, if it has one
            appliance_activity = appliances_data.loc[device_type]['activity']
            if appliance_activity == "LEVEL" or appliance_activity == "ACTIVE_OCC" or
            appliance_activity == "CUSTOM":
                activity_profile = 0
            else:
                activity_profile = self.activity_profiles.xs(appliance_activity, level='activity')
            # initialize the appliance
            new_appliance = appliance_object.Appliance(device_type,
            appliances_data.loc[device_type].to_dict(), \
            activity_profile, self.relative_monthly_temperatures, self.constants.loc["calibration"].iloc[0])
            assigned_appliances.append(new_appliance)
    return (assigned_appliances, self.get_appliance_names(assigned_appliances)) # return a tuple

# get the used appliance names
def get_appliance_names(self, assigned_appliances):
    assigned_appliance_names = []
    for appliance in assigned_appliances:
        assigned_appliance_names.append(appliance.name)
    return assigned_appliance_names

# take the base_appliances dataframe as input (load from csv), calculate the new values per appliance via
get_new_appliance_data
# then merge the resulting dataframe with the initial to create the complete appliance dataframe
def complete_data(self):
    appliances = shared.load_dataframe_from_excel(self.base_appliances_data_path, 'appliances')
    new_data = []
    for appliance in appliances.index.tolist():
        new_data.append(self.get_new_appliance_data(appliances.loc[appliance]))
    new_dataframe = pandas.DataFrame(new_data, index=appliances.index.tolist(),
    columns=["calibrated_yearly_cycles", \
    "restart_delay", "mean_cycle_energy_demand", "yearly_on_time", "yearly_off_time", \
    "proportion_of_start_ups_due_to_occupancy", "yearly_minutes_appliance_can_start", \
    "yearly_mean_time_between_starts", "lamda", "calibration_scalar", \
    "yearly_standby_energy_use", \

```

```

        "total_yearly_demand",                                "yearly_on_energy_use",
"overall_average_per_dwelling_including_ownership_data")]
complete_dataframe      =      pandas.concat([appliances,      new_dataframe],      axis=1,
join_axes=[appliances.index])
return complete_dataframe

def get_new_appliance_data(self, appliance):
    # just call all the functions calculating the needed values, append results to a list and return that list
    new_values = []
    calibrated_yearly_cycles = self.get_calibrated_yearly_cycles(appliance)
    ten_min_cycle_length = self.get_ten_min_cycle_length(appliance)
    restart_delay = self.get_restart_delay(appliance)
    mean_cycle_energy_demand      =      self.get_mean_ten_min_cycle_energy_demand(appliance,
ten_min_cycle_length)
    yearly_on_time = self.get_yearly_on_time(appliance, calibrated_yearly_cycles)
    yearly_off_time = self.get_yearly_off_time(yearly_on_time)
    proportion_of_start_ups_due_to_occupancy      =
self.get_proportion_of_start_ups_due_to_occupancy(appliance)
    yearly_minutes_appliance_can_start      =      self.get_yearly_minutes_appliance_can_start(appliance,
calibrated_yearly_cycles, \
                           restart_delay,                  proportion_of_start_ups_due_to_occupancy,
yearly_on_time)

    yearly_mean_time_between_starts      =
self.get_yearly_mean_time_between_starts(calibrated_yearly_cycles, \
                           yearly_minutes_appliance_can_start)
    lamda = self.get_lamda(yearly_mean_time_between_starts)
    calibration_scalar = self.get_calibration_scalar(appliance, lamda)
    yearly_on_energy_use      =      self.get_yearly_on_energy_use(calibrated_yearly_cycles,
mean_cycle_energy_demand)
    yearly_standby_energy_use = self.get_yearly_standby_energy_use(appliance, yearly_on_time)
    total_yearly_demand      =      self.get_total_yearly_demand(yearly_on_energy_use,
yearly_standby_energy_use)
    overall_ave_per_dwelling_incl_ownership_data      =
self.get_overall_ave_per_dwelling_incl_ownership_data(appliance, total_yearly_demand)

    new_values.append(calibrated_yearly_cycles)
    new_values.append(restart_delay)
    new_values.append(mean_cycle_energy_demand)
    new_values.append(yearly_on_time)
    new_values.append(yearly_off_time)
    new_values.append(proportion_of_start_ups_due_to_occupancy)
    new_values.append(yearly_minutes_appliance_can_start)
    new_values.append(yearly_mean_time_between_starts)
    new_values.append(lamda)
    new_values.append(calibration_scalar)
    new_values.append(yearly_standby_energy_use)
    new_values.append(total_yearly_demand)
    new_values.append(yearly_on_energy_use)
    new_values.append(overall_ave_per_dwelling_incl_ownership_data)
    return new_values

```

```

# below are all the functions called from the get_new_appliance_data function
def get_calibrated_yearly_cycles(self, appliance):
    return self.constants.loc["calibration"]*appliance["base_yearly_cycles"]

def get_ten_min_cycle_length(self, appliance):
    new_length = round(appliance["mean_cycle_duration"]/10)
    if new_length == 0:
        return 1
    else:
        return new_length

def get_ten_min_restart_delay(self, appliance, ten_min_cycle_length):
    return ten_min_cycle_length*appliance["restart_delay_multiplier"]

def get_mean_ten_min_cycle_energy_demand(self, appliance, ten_min_cycle_length):
    power_modifier = (appliance["mean_cycle_duration"]/(10*ten_min_cycle_length))
    return power_modifier*appliance["mean_cycle_power"]

def get_yearly_on_time(self, appliance, calibrated_yearly_cycles):
    return appliance["mean_cycle_duration"]*calibrated_yearly_cycles

def get_yearly_off_time(self, yearly_on_time):
    return 365*24*60 - yearly_on_time

def get_proportion_of_start_ups_due_to_occupancy(self, appliance):
    if appliance["active_occupancy_dependent"] == "Yes":
        return self.constants.loc["mean_active_occupancy"]
    else: return 1

def get_yearly_minutes_appliance_can_start(self, appliance, calibrated_yearly_cycles, \
                                         restart_delay, proportion_of_start_ups_due_to_occupancy, yearly_on_time):
    return (365*24*60*proportion_of_start_ups_due_to_occupancy - (yearly_on_time + \
calibrated_yearly_cycles * restart_delay))

def get_yearly_mean_time_between_starts(self, calibrated_yearly_cycles, \
                                         yearly_minutes_appliance_can_start):
    return yearly_minutes_appliance_can_start/calibrated_yearly_cycles

def get_lamda(self, yearly_mean_time_between_starts):
    return 1/yearly_mean_time_between_starts

def get_calibration_scalar(self, appliance, lamda):
    return lamda/appliance["average_activity_probability"]

def get_yearly_on_energy_use(self, calibrated_yearly_cycles, mean_cycle_energy_demand):
    return calibrated_yearly_cycles*mean_cycle_energy_demand

def get_yearly_standby_energy_use(self, appliance, yearly_on_time):
    return (365*24*60-yearly_on_time)*(appliance["standby_power"]/(60*1000))

def get_total_yearly_demand(self, yearly_on_energy_use, yearly_standby_energy_use):
    return yearly_on_energy_use + yearly_standby_energy_use

```

```
def get_overall_ave_per_dwelling_incl_ownership_data (self, appliance, total_yearly_demand):
    return total_yearly_demand*appliance["possibility_to_get"]
```

8.1.5. Luminaire_data

```
import numpy
import pandas
import luminaire
import shared

class LuminaireData:
    """methods to assign luminaires to a house and calculate a house's irradiance threshold"""

    def __init__(self, luminaires_datapath, constants_datapath, durations_datapath):
        self.luminaire_data = shared.load_dataframe_from_csv(luminaires_datapath)
        self.constants = shared.load_dataframe_from_csv(constants_datapath)
        self.duration_data = shared.load_dataframe_from_csv(durations_datapath)

    # assign new luminaires to a house
    # choose a house type from the samples, then create a luminaire object for each bulb
    def assign_new_luminaires(self):
        house_luminaires = []
        for item in self.luminaire_data.loc[numpy.random.choice(self.luminaire_data.index.tolist())]:
            # filter out the NaN values
            if not numpy.isnan(item):
                house_luminaires.append(luminaire.Luminaire(item,
                    self.constants.at["calibration_scalar","value"], self.duration_data))
        return house_luminaires

    def get_house_irradiance_threshold(self):
        return numpy.random.normal(self.constants.at["global_irradiance_threshold","value"],
            self.constants.at["irradiance_deviation","value"])
```

8.1.6. Appliance

```
import numpy
import shared

# This module implements the Appliance class
# An appliance is an object that a house can have, and it has many attributes relative to power demand.
# Data for an appliance is drawn from the appliances.csv file
# This module also contains methods to get the current_power_drain of an appliance, depending on the
# time, day and occupancy

class Appliance:
    """This is a house appliance"""

    def __init__(self, device_type, appliance_data, activity_profile, relative_monthly_temperatures,
        calibration):
        self.device_type = device_type
```

```

# The one-liner below uses the appliance_data dictionary to dynamically get and set the attributes this
appliance has
for attribute, value in appliance_data.items(): setattr(self, attribute, value)
self.cycle_time_left = 0
self.restart_delay_time_left = self.vary_restart_delay_time_left() # random time left for end of the
cycle
self.current_power_drain = self.standby_power # appliances start in standby
self.activity_profile = activity_profile # this is 0 if there is no specific profile and a dataframe
otherwise
self.relative_monthly_temperatures = relative_monthly_temperatures
self.calibration = calibration

def vary_restart_delay_time_left(self):
    return self.restart_delay * numpy.random.random() * 2 # following the CREST model, weight is 2 to
add diversity

# vary power drain using a normal distribution. mean_cycle_power is center and mean_cycle_power/10
is the standard deviation
def vary_mean_cycle_power(self):
    if self.mean_cycle_power == 0.0 : return 0.0
    return numpy.random.normal(self.mean_cycle_power, self.mean_cycle_power/10.0)

# vary the cycle length of appliance with a normal distribution
def vary_mean_cycle_duration(self):
    return numpy.random.normal(self.mean_cycle_duration, self.mean_cycle_duration/10)

# The average viewing time is approximately 73 minutes, function based on CREST implementation
def vary_tv_viewing_time(self):
    return int(round(70*numpy.power(0-numpy.log10(1-numpy.random.random()), 1.1)))

def decrement_delay_time_left(self):
    self.restart_delay_time_left -= 10

def decrement_cycle_time_left(self):
    self.cycle_time_left -= 10

def switch_on(self):
    # if its cycle lasts 10' or more, proceed normally
    if self.get_cycle_length() > 9:
        self.cycle_time_left = self.get_cycle_length()
        self.restart_delay_time_left = self.restart_delay
        self.current_power_drain = self.get_current_power_usage()
        self.decrement_cycle_time_left()
    # if its a small cycle appliance, power drain will be scaled. Also, it will *never* really switch on.
    else:
        self.current_power_drain = self.get_current_power_usage()*(self.mean_cycle_duration/10.0)

def switch_off(self):
    self.current_power_drain = self.standby_power

def get_cycle_length(self):
    if self.device_type[:2] == 'TV': # this is for all TV types

```

```

        return self.vary_tv_viewing_time()
    elif (self.device_type == 'STORAGE_HEATER') or (self.device_type == 'ELEC_SPACE_HEATING'):
        return self.vary_mean_cycle_duration()
    else:
        return self.mean_cycle_duration

    def get_current_power_usage(self):
        # For washing machines, the power_draw_at_cycle_time list contains expected power draw at specific times
        # during the cycle (index is the cycle time). This models an example power profile for an example washing machine,
        # based upon data from personal communication with a major washing machine manufacturer
        if ((self.name == 'WASHING_MACHINE') or (self.name == 'WASHER_DRYER')):
            return self.ten_min_power_draw_at_cycle_time[self.mean_cycle_duration - self.cycle_time_left]
        else:
            return self.vary_mean_cycle_power() # power should vary little each time, based on the CREST model

    def get_appliance_switch_on_probability(self, ten_minute_period, day, weekend, active_occupancy):
        if (self.activity != 'LEVEL') and (self.activity != 'ACTIVE_OCC'): # for appliances depending on activity profile
            return self.activity_profile.ix[str((weekend, active_occupancy)),ten_minute_period-1]
        elif self.device_type == 'ELEC_SPACE_HEATING': # space heater startup depends on month
            return self.relative_monthly_temperatures.iat[shared.get_month(day)-1,0]
        else:
            return 1

    def get_storage_heater_switch_on_probability(self, ten_minute_period, day): # storage heaters have a simple representation
        month = shared.get_month(day)
        if ten_minute_period == 4: # will only start between 00:30 and 00:40
            # we assume that the coldest day is Jan 14.
            # so the start day is calibrated_yearly_cycles/2 days before, and the stop day is calibrated_yearly_cycles/2 days after
            date_on = int(round(14 - self.calibrated_yearly_cycles/2))
            if date_on < 1: date_on += 365
            month_on = shared.get_month(date_on)

            date_off = int(round(14 + self.calibrated_yearly_cycles/2))
            if date_off > 365: date_off -= 365
            month_off = shared.get_month(date_off)

            if (month_on > month > month_off): # it is summer
                appliance_switch_on_probability = 0
            elif (month_on < month) or (month < month_off): # its winter
                appliance_switch_on_probability = 1
            else: # for poweron and poweroff months, chance to start is 50% per 10_min period
                appliance_switch_on_probability = 0.5
            return appliance_switch_on_probability
        return 0.0 # appliance cannot start if the if loop didn't run

```

```

def handle_appliances_turned_off(self, ten_minute_period, day, weekend, active_occupancy):
    # There must be active occupants, or the profile must not depend on occupancy for a start event to occur
    if (self.activity == 'LEVEL') or (self.activity != 'CUSTOM' and active_occupancy > 0):
        appliance_switch_on_probability = self.get_appliance_switch_on_probability(ten_minute_period,
        day, weekend, active_occupancy)
        # determine a possible switch_on
        if numpy.random.random() < self.calibration * appliance_switch_on_probability: # not sure why we need to calibrate it
            self.switch_on()
    elif self.device_type == 'STORAGE_HEATER': # storage heaters have a simple representation.
        appliance_switch_on_probability = self.get_storage_heater_switch_on_probability(ten_minute_period, day)
        # determine a possible switch_on
        if numpy.random.random() < appliance_switch_on_probability:
            self.switch_on()

def get_power_usage(self, ten_minute_period, day, active_occupancy):
    if self.cycle_time_left < 1: # if appliance is off
        if self.restart_delay_time_left > 0: #if it is in cooldown
            self.decrement_delay_time_left()
        else: # if appliance is ready to start again
            self.handle_appliances_turned_off(ten_minute_period, day, shared.get_weekend(day),
            active_occupancy)
    else: # if the appliance is on
        # if there is still a resident active, or the appliance does not need occupancy to be active
        if (active_occupancy != 0) or (self.activity == 'LEVEL') or (self.activity == 'ACT_LAUNDRY') or (self.activity == 'CUSTOM'):
            if self.cycle_time_left < 10: # if the appliance is about to switch off inside this 10min period, scale down the power drain
                self.current_power_drain = self.current_power_drain*(self.cycle_time_left/10.0)
                self.decrement_cycle_time_left()
            else: # if there are no active residents, but the appliance is dependent on active residents
                self.switch_off()
            # The activity will be completed upon the return of the active occupancy. so no need to touch the cycle_time_left
        # handle appliances finishing after this cycle
        if self.cycle_time_left < 1:
            power_drain_this_period = self.current_power_drain
            self.switch_off()
            return power_drain_this_period
        else:
            return self.current_power_drain

```

8.1.7. Luminaire

```

import numpy
import pandas
import shared

class Luminaire:
    """This is a lighting unit"""

```

```

def __init__(self, power_rating, calibration_scalar, duration_data, tech=""):
    self.tech=tech # in the future it could be one of the usual types - CFL, LED etc.
    self.power_rating = power_rating
    self.duration_data = duration_data
    self.calibrated_relative_use = -calibration_scalar*numpy.log(numpy.random.random()) # relative use
    varies between ln(0+)~ln(1-). This simulates different use profiles for lamps
    self.current_power = 0 # assume lamps are off initially
    self.cycle_time_left = 0

    # return the probability of a switch-on due to irradiance
    # there is a 5% chance that probability will be 1 even if there is enough light
    def get_light_level_switch_on_chance(self, ten_minute_period, irradiance, irradiance_threshold):
        if (irradiance.iat[ten_minute_period*10-1] < irradiance_threshold) or (numpy.random.random() <
0.05): return 1
        else: return 0

    # parse the duration_data array to determine the on_duration
    def get_on_duration(self):
        # for each period type
        for next_period in self.duration_data.values:
            # if this period "happens"
            if next_period[2] > numpy.random.random():
                # return the exact minutes the lamp will stay on as a random between the low&high as defined in
                the period's array
                return numpy.random.randint(next_period[0], next_period[1]+1)

    def decrement_cycle_time(self):
        self.cycle_time_left -= 10

    def switch_on(self):
        self.cycle_time_left = self.get_on_duration()
        # handle low duration cycles - scale down the power drain
        if self.cycle_time_left < 11:
            self.current_power = self.power_rating*(self.cycle_time_left/10.0)
        else:
            self.current_power = self.power_rating
        self.decrement_cycle_time()

    def switch_off(self):
        self.current_power = 0

    def get_power_usage(self, ten_minute_period, day, effective_active_occupancy, irradiance,
irradiance_threshold):
        # if the luminaire was off, determine possible switch-ons
        if self.current_power == 0:
            light_level_condition_test = self.get_light_level_switch_on_chance(ten_minute_period, irradiance,
irradiance_threshold)
            switch_on_chance
            light_level_condition_test*self.calibrated_relative_use*effective_active_occupancy
            if switch_on_chance > numpy.random.random():
                self.switch_on()
        # if it was on, determine possible switch-offs

```

```

else:
    if self.cycle_time_left < 1 or effective_active_occupancy == 0:
        self.switch_off()
    # if the light is at the end of the cycle, scale down its power drain
    elif self.cycle_time_left < 10:
        self.current_power = self.current_power*(self.cycle_time_left/10.0)
        self.decrement_cycle_time() # if it was on, always decrement cycle time
    return self.current_power

```

8.1.8. Electricity

```

import numpy
import pandas
import shared

# This module implements the electricity class and is responsible for getting power demand for a given duration
# Its inputs are assigned_appliances_data from the house's appliances,
# active_occupancy_data from the occupancy module, and a start_day and specific duration.
# It turn it responds with electricity demand data for every minute of every day specified, starting at 00:00
# The data returned is in a dataframe format, with rows being the house appliances and
# columns being a multiindex with the day as level 1 and the minute as level 2

class Electricity:
    """ all methods for determining electricity demand, excluding lighting"""

    def get_daily_electricity_demand(self, day, assigned_appliances_data, daily_active_occupancy):
        minute_demand = []
        for ten_minute_period in range(1,145):
            active_occupancy_this_minute = daily_active_occupancy.iat[ten_minute_period-1, 0]
            power_demand = []
            for appliance in assigned_appliances_data[0]:
                power_demand.append(appliance.get_power_usage(ten_minute_period, day,
                                                active_occupancy_this_minute))
            minute_demand.append(pandas.Series(power_demand, index = assigned_appliances_data[1]))
        return pandas.concat(minute_demand, axis=1, keys = numpy.arange(1,145)) # return in dataframe format inside list

        # this calls the get_daily_electricity_demand to update the power_consumption array for the days included in the days list

    def electricity_data_updater(self, days, assigned_appliances_data, active_occupancy_data):
        daily_demand = []
        for day in days:
            # each day's occupancy is based on 12:00 of the day before, so previous_active_occupancy is occupancy_data[-1]
            daily_demand.append(self.get_daily_electricity_demand(day, assigned_appliances_data,
                                                active_occupancy_data.loc[day]))
        return pandas.concat(daily_demand, axis=1, keys = days)

    def get_electricity_demand(self, assigned_appliances_data, active_occupancy_data, start_day, duration):
        # run the simulation

```

```

        return self.electricity_data_updater(numpy.arange(start_day, duration+start_day),
assigned_appliances_data, \
                                         active_occupancy_data)

```

8.1.9. Lighting

```

import numpy
import pandas
import shared

class Lighting:
    """This is the lighting power demand module"""

    def __init__(self, irradiance_datapath, effective_occupancy_datapath):
        self.irradiance_data = shared.load_dataframe_from_csv(irradiance_datapath)
        self.effective_occupancy_data = shared.load_dataframe_from_csv(effective_occupancy_datapath)

    # effective occupancy is an effect of light sharing between occupants
    def get_effective_occupancy(self, active_occupancy_this_minute):
        return self.effective_occupancy_data.iat[active_occupancy_this_minute,0]

    def get_daily_lighting_demand(self, day, assigned_luminaires, daily_active_occupancy,
irradiance_threshold):
        irradiance = self.irradiance_data[str(shared.get_month(day))] # irradiance data is available per minute
for each month
        minute_demand=[]
        for ten_minute_period in range (1,145):
            active_occupancy_this_minute = daily_active_occupancy.iat[ten_minute_period-1, 0]
            effective_occupancy_this_minute = self.get_effective_occupancy(active_occupancy_this_minute)
            power_demand = []
            for item in assigned_luminaires:
                power_demand.append(item.get_power_usage(ten_minute_period,
day,
effective_occupancy_this_minute, \
                                         irradiance, irradiance_threshold))
            minute_demand.append(pandas.Series(power_demand))
        return pandas.concat(minute_demand, axis=1, keys = numpy.arange(1,145)) # return in dataframe
format inside list

    # this calls the get_daily_electricity_demand to update the power_consumption array for the days
included in the days list
    def lighting_data_updater(self, days, assigned_luminaires, active_occupancy_data,
irradiance_threshold):
        daily_demand=[]
        for day in days:
            daily_demand.append(self.get_daily_lighting_demand(day,
assigned_luminaires,
active_occupancy_data.loc[day], irradiance_threshold))
        return pandas.concat(daily_demand, axis=1, keys = days)

    def get_lighting_demand(self, assigned_luminaires, active_occupancy_data, start_day, duration,
irradiance_threshold):
        return self.lighting_data_updater(numpy.arange(start_day, duration+start_day), assigned_luminaires, \
                                         active_occupancy_data, irradiance_threshold)

```

8.1.10. Shared

```
import numpy
import pandas
import json

# This module is a library containing various functions used by other modules

# ***I/O functions***
# load
def load_dataframe_from_csv(path, index_col=0, header='infer'):
    return pandas.read_csv(path, header=header, index_col=index_col)

def load_dataframe_from_excel(path, sheet, header=9, index=[0]):
    return pandas.read_excel(path, sheetname=sheet, header=header, index_col=index)

def load_json(path):
    with open(path) as json_data:
        json_data = json.load(json_data)
    return json_data

# save
def save_dataframe_to_csv(dataframe, path, index_label):
    dataframe.to_csv(path, index_label=index_label)

def save_dataframe_to_excel(dataframe, path, sheet, startrow=9):
    dataframe.to_excel(path, sheet, index_label='label', merge_cells=False, startrow=startrow)

def write_json(json_data, path):
    with open(path, "w") as new_json:
        json.dump(json_data, new_json, sort_keys=True, indent=4) # dump in human-readable format to get pretty JSON

# ***Conversion functions***
# convert a dataframe to a hierarchical dictionary, the way we need it for occupancy indexing
def dataframe_to_dict(dataframe):
    newdict = {}
    for index in dataframe.index.tolist():
        newdict.update({index : dataframe.loc[index].values.tolist()})
    return newdict

# convert a multiindex dataframe to single_index for faster index searches
def convert_multiindex_to_singleindex(dataframe):
    data = dataframe.values
    index_name = ','.join(dataframe.index.names)
    old_index = dataframe.index.tolist()
    new_index=[]
    for item in old_index:
        new_index.append(str(tuple(item)))
    return pandas.DataFrame(data, index=pandas.Index(new_index), name=index_name),
           columns=dataframe.columns)
```