ΕΘΝΙΚΟΝ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟΝ
ΠΑΝΕΠΙΣΤΗΜΙΟΝ ΑΘΗΝΩΝ

Σχολή Θετικών Επιστημών, Τμήμα Φυσικής

Μεταπτυχιακό Πρόγραμμα Ηλεκτρονικού Αυτοματισμού

Ανίχνευση Συμβάντος σε Πολυδιάστατα Δεδομένα

Καλβουρίδη Ειρήνη

Αριθμός Μητρώου: 2011 503
e-mail: eri.kalvouridi@gmail.com

Επιβλέπων Καθηγητής:
Χατζηευθυμιάδης Ευστάθιος
Αναπληρωτής Καθηγητής ΕΚΠΑ

Αθήνα 2016

Περίληψη

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του Διατμηματικού Προγράμματος Μεταπτυχιακών Σπουδών (ΔΠΜΣ) με τον τίτλο ειδίκευσης «*Ηλεκτρονικός Αυτοματισμός*», το οποίο λειτουργεί υπό τη συνεργασία των τμημάτων Φυσικής και Πληροφορικής του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών.

Το αντικείμενο μελέτης της εργασίας αυτής, αφορά την εφαρμογή αλγορίθμων ανίχνευσης συμβάντος (event detection algorithms). Οι αλγόριθμοι που μελετήθηκαν και υλοποιήθηκαν είναι ο μονόπλευρος αλγόριθμος Tabular CUSUM για μία μεταβλητή και ο μονόπλευρος αλγόριθμος CUSUM για δύο μεταβλητές, όπως περιγράφεται από τον R. B. Crosier. Η εφαρμογή των αλγορίθμων αυτών έγινε πάνω στην πλατφόρμα SunSPOT και ο κώδικας στον οποίο αναπτύχθηκαν οι εφαρμογές συντάχθηκε στη γλώσσα Java. Το περιεχόμενο της εργασίας έχει οργανωθεί σε έξι κεφάλαια και συγκεκριμένα:

- Το **Κεφάλαιο 1** είναι μια εισαγωγή, στην οποία γίνεται μια ιστορική αναδρομή σχετικά με την εξέλιξη της τεχνολογίας και το πώς η εξέλιξη αυτή έχει επηρεάσει προβλήματα που αφορούν την ανίχνευση συμβάντος.

- Στο **Κεφάλαιο 2** γίνεται μια σύντομη αναφορά στον *Έλεγχο Ποιότητας*, έναν από τους πρώτους τομείς στον οποίο διερευνήθηκε η ανίχνευση συμβάντος και εφαρμόστηκαν για πρώτη φορά τα διαγράμματα ελέγχου.

- Το **Κεφάλαιο 3** παρουσιάζει το *Στατιστικό Έλεγχο Ποιότητας*, όπου περιγράφονται κάποιες βασικές στατιστικές έννοιες που χρησιμοποιούνται για τον έλεγχο διάφορων διεργασιών.

- Στο **Κεφάλαιο 4** γίνεται μια αναλυτική περιγραφή των *Διαγραμμάτων Ελέγχου Shewhart*.

- Στο **Κεφάλαιο 5** περιγράφονται τα *Διαγράμματα Ελέγχου CUSUM* για μία μεταβλητή και για πολλές μεταβλητές.

- Στο **Κεφάλαιο 6** γίνεται αρχικά μια σύντομη περιγραφή της πλατφόρμας SunSPOT, αναλύονται οι εφαρμογές που υλοποιήθηκαν και δίνεται ο κώδικας που συντάχθηκε για την καθεμία σε γλώσσα Java.

Λέξεις κλειδιά: ανίχνευση συμβάντος, διαγράμματα ελέγχου Shewhart, διαγράμματα CUSUM, MCUSUM, στατιστική του Hotelling, Crosier

Abstract

The present master thesis was written as part of the joint/inter-disciplinary M.Sc. program entitled as "Master in Electronic Automation" that runs by the department of Physics and the department of Informatics and Telecommunications of the National and Kapodistrian University of Athens.

The subject of this thesis is the application of event detection algorithms. The algorithms that were studied and developed are the one-sided univariate Tabular CUSUM algorithm and the one-sided bivariate CUSUM algorithm as described by R. B. Crosier. Those algorithms were applied on the SunSPOT platform and were developed in Java. The context of the thesis is organized in six chapters, namely:

- Chapter 1 is an introduction, where we discuss the evolution of technology and how this extraordinary development has affected event detection problems.
- Chapter 2 is a brief reference to Quality Control, where the event detection was firstly investigated and applied to with the use of the emerging control charts.
- Chapter 3 presents the Statistical Quality Control, where we describe some of the basic statistical concepts used for the Statistical Process Control (SPC).
- Chapter 4 gives a detailed description of the Shewhart control charts.
- Chapter 5 describes the univariate and multivariate CUSUM control charts.
- Finally, in Chapter 6 after a brief description of the SunSPOT platform, we analyze the applications that were implemented and we present their respective codes that were developed in Java.

Keywords: event detection, Shewhart control charts, CUSUM control charts, MCUSUM, Hotelling's T^2-Statistic, Crosier

Περιεχόμενα

Κατάλογος Σχημάτων

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ.Ε.Χατζηευθυμιάδη για την υπόδειξη του θέματος της διπλωματικής μου και για τη συνεχή καθοδήγησή του κατά τη διάρκεια εκπόνησής της.

1 Εισαγωγή

Αν το πρώτο μισό του $20^{ου}$ αιώνα χαρακτηρίστηκε από τη μεγάλη βιομηχανική επανάσταση και από την εμφάνιση των δύο μεγάλων θεωριών της Φυσικής, της θεωρίας της Σχετικότητας και της Κβαντομηχανικής, που άλλαξαν κυριολεκτικά την πορεία της Επιστήμης, το δεύτερο μισό του αιώνα αυτού χαρακτηρίστηκε από τη ραγδαία εξέλιξη των ηλεκτρονικών υπολογιστών και τη μεγάλη ανάπτυξη της τεχνολογίας που συνετέλεσαν μεταξύ των άλλων και στην κατάκτηση του Διαστήματος.

Ειδικότερα οι ψηφιακοί υπολογιστές (digital computers) κατέστησαν δυνατά πολλά επιστημονικά και βιομηχανικά επιτεύγματα και άνοιξαν νέους ορίζοντες σε όλους τους τομείς της ανθρώπινης δραστηριότητας. Με τις τεράστιες υπολογιστικές δυνατότητες που συνεχώς διευρύνονται, και την ικανότητα προσαρμογής τους σε κάθε επιστημονικό ή τεχνολογικό περιβάλλον, συνέβαλαν και συμβάλλουν στην πρόοδο και την ανάπτυξη της ανθρωπότητας δημιουργώντας ταυτόχρονα νέες εφαρμογές και απεριόριστες προοπτικές για το μέλλον.

Με την ανάπτυξη της Ηλεκτρονικής, της Μικροηλεκτρονικής και της Πληροφορικής έχουν επιτευχθεί αξιόλογα επίπεδα σμίκρυνσης (micronization), τα οποία έχουν επιτρέψει την ύπαρξη σημαντικής υπολογιστικής ισχύος σε ολοένα και μικρότερες συσκευές. Πλέον έχουμε στη διάθεσή μας έξυπνους αισθητήρες (smart sensors) με πιο ισχυρούς επεξεργαστές και περισσότερη μνήμη από ότι είχαν οι προσωπικοί υπολογιστές παλαιότερων γενεών, και αυτό σε πολύ μικρές διαστάσεις συγκρίσιμες με ένα σπιρτόκουτο. Οι αισθητήρες αυτοί μπορούν να υλοποιούν πολύπλοκα πρωτόκολλα, να επεξεργάζονται οι ίδιοι μερικώς ή ολικώς τις μετρήσεις που λαμβάνουν από το περιβάλλον, έως και να φιλοξενούν ένα περιορισμένο λειτουργικό σύστημα.

Αυτή η μεγάλη ανάπτυξη που έχει σημειωθεί στην τεχνολογία έχει επιφέρει μια ανάλογη αύξηση στην προσπάθεια επίλυσης πραγματικών προβλημάτων, τα οποία αφορούν θέματα όπως:

- Την ανίχνευση και διάγνωση βλάβης (monitoring)
- Τη διατήρηση κατάστασης βιομηχανικών διεργασιών
- Την ασφάλεια πολύπλοκων συστημάτων (αεροσκάφη, πλοία, πύραυλοι κ.λπ.)
- Τον έλεγχο ποιότητας (Quality Control)
- Την πρόβλεψη των φυσικών καταστροφών (σεισμοί, τσουνάμι, κ.λπ.)
- Την παρακολούθηση διεργασιών στη βιοϊατρική, κ.λπ.

Τα προβλήματα αυτά έχουν προκύψει κυρίως λόγω της αυξανόμενης πολυπλοκότητας των περισσότερων τεχνολογικών διαδικασιών καθώς και της ύπαρξης εξελιγμένων πληροφοριακών συστημάτων επεξεργασίας δεδομένων. Λύσεις στα προβλήματα αυτά είναι ζωτικής σημασίας για την ανθρώπινη ασφάλεια, την οικολογία και την οικονομία. Μία σημαντική συμβολή στη λύση των προβλημάτων αυτών έχουν προσφέρει τα προαναφερόμενα πληροφοριακά συστήματα επεξεργασίας δεδομένων. Με την εξέλιξη που έχουν παρουσιάσει έχουν καταστήσει δυνατή την ανάπτυξη, τη μελέτη και την εφαρμογή πολύπλοκων αλγορίθμων παρακολούθησης των διαφόρων διεργασιών.

Το κοινό χαρακτηριστικό των παραπάνω προβλημάτων είναι το γεγονός ότι η επίλυση του καθενός στηρίζεται στην ανίχνευση μίας ή περισσοτέρων απότομων αλλαγών που παρουσιάζονται σε ορισμένες χαρακτηριστικές ιδιότητες της εξεταζόμενης διεργασίας. Ως απότομη αλλαγή σε μία παράμετρο (μεταβλητή) ενός συστήματος ονομάζουμε μια οποιαδήποτε αλλαγή στην παράμετρο αυτή, η οποία γίνεται είτε άμεσα είτε πολύ γρήγορα σε σχέση με την περίοδο δειγματοληψίας των μετρήσεων.

Το μέγεθος της αλλαγής στην παράμετρο που εξετάζεται δεν παίζει ρόλο στο να χαρακτηριστεί ως απότομη αυτή η αλλαγή. Στις περισσότερες περιπτώσεις το κύριο πρόβλημα που τίθεται προς επίλυση αφορά την ανίχνευση μικρών αλλαγών που θα παρουσιαστούν σε μια χαρακτηριστική ιδιότητα, και αυτό προκύπτει επειδή η συσσώρευση μικρών αλλαγών σε αυτές τις περιπτώσεις μπορεί να έχει καταστροφικές συνέπειες.

Για παράδειγμα, τα μικρά ελαττώματα που μπορεί να προκύψουν στους αισθητήρες ενός συστήματος πλοήγησης μπορεί να οδηγήσουν σε σοβαρά σφάλματα κατά την εκτίμηση της θέσης ενός αεροπλάνου. Ένα άλλο παράδειγμα είναι η έγκαιρη προειδοποίηση μικρών αποκλίσεων από τις κανονικές συνθήκες λειτουργίας μιας βιομηχανικής διεργασίας. Σε αυτές τις περιπτώσεις, η έγκαιρη ανίχνευση μικρών αλλαγών επιτρέπει το σχεδιασμό κατάλληλου προγράμματος παρακολούθησης, επιθεώρησης και ενδεχομένως επισκευής (διόρθωσης) της συγκεκριμένης διεργασίας, ώστε να αποφεύγεται και η αύξηση του κόστους.

Η αλλαγή, λοιπόν, όταν αναφερόμαστε σε μία διεργασία ή σε ένα σύστημα, ανεξάρτητα από το μέγεθός της μπορούμε να πούμε ότι είναι η μετάβαση από μία κατάσταση σε μία άλλη. Το πρόβλημα της ανίχνευσης αυτών των αλλαγών έχει διερευνηθεί εκτενώς από στατιστικούς και μαθηματικούς και είναι γνωστό ως *ανίχνευση συμβάντος* (event detection, change detection). Μία από τις πρώτες εφαρμογές της ανίχνευσης συμβάντος ήταν ο *Έλεγχος Ποιότητας* κατά την συνεχή παρακολούθηση της παραγωγής προϊόντων (Shewhart, 1920). Οι τεχνικές ανίχνευσης αλλαγών που αναπτύχθηκαν κατά τον Έλεγχο Ποιότητας χρησιμοποιήθηκαν και σε άλλους τομείς και εξελίχθηκαν από διάφορους επιστήμονες ανάλογα με το πρόβλημα που έπρεπε να λυθεί.

2 Έλεγχος Ποιότητας

Η έννοια της ποιότητας υπάρχει εδώ και πολλά χρόνια, αν και το νόημά της έχει αλλάξει και έχει εξελιχθεί με την πάροδο του χρόνου. Στις αρχές του εικοστού αιώνα, η διαχείριση της ποιότητας σήμαινε τον έλεγχο των προϊόντων ώστε να εξασφαλίζεται ότι πληρούν τις προκαθορισμένες προδιαγραφές. Στη δεκαετία του 1940, κατά τη διάρκεια του δεύτερου Παγκοσμίου Πολέμου, η ποιότητα άρχισε να μελετάται περισσότερο με τη βοήθεια της Στατιστικής (Στατιστικός Έλεγχος Ποιότητας). Στη δεκαετία του 1960, η ποιότητα πήρε μια σχετικά πιο ευρεία σημασία και άρχισε να καλύπτει ολόκληρη την επιχείρηση και όχι μόνο τη διαδικασία της παραγωγής. Προς το τέλος του 1970, η έννοια της ποιότητας για τις επιχειρήσεις άλλαξε καθοριστικά. Η συνεχόμενη αύξηση του ανταγωνισμού μεταξύ των εταιρειών τοποθέτησε την ποιότητα σε πρωταγωνιστικό ρόλο. Οι εταιρείες άρχισαν να εστιάζουν πλέον στη βελτίωση της ποιότητας, προκειμένου να είναι όλο και πιο ανταγωνιστικές και να καταφέρουν να επιβιώσουν στην αγορά.

Πολλές εταιρείες προσέλαβαν συμβούλους ποιότητας και ανέπτυξαν προγράμματα κατάρτισης για τους εργαζομένους τους, αποκτώντας έτσι η ποιότητα μια πιο στρατηγική σημασία. Οι επιτυχημένες εταιρείες άρχισαν να κατανοούν ότι η επένδυση στη συνεχόμενη βελτίωση της ποιότητας τους παρείχε ένα ανταγωνιστικό πλεονέκτημα. Πρωταρχικό τους μέλημα ήταν οι ανάγκες του πελάτη, με βάση τις οποίες έπρεπε πλέον να καθορίζεται η ποιότητα, ώστε να ικανοποιεί ή και να υπερβαίνει τις προσδοκίες του.

2.1 Η Έννοια της Ποιότητας

Η ποιότητα είναι μια έννοια σύνθετη, για την οποία έχουν δοθεί πολλοί και διαφορετικοί ορισμοί. Οι περισσότεροι άνθρωποι έχουν μια διαισθητική άποψη για το τι είναι ποιότητα και τη συνδέουν με τα επιθυμητά χαρακτηριστικά που πρέπει να έχει ένα προϊόν ή μια υπηρεσία. Ένας παραδοσιακός ορισμός για την έννοια της ποιότητας έχει δοθεί από τον Joseph

Juran, σύμφωνα με τον οποίο:

> «Ποιότητα είναι η προσαρμογή των χαρακτηριστικών ενός προϊόντος ή μιας υπηρεσίας στις απαιτήσεις του καταναλωτή (χρήστη), δηλαδή η έννοια ποιότητα σημαίνει καταλληλότητα προς χρήση.»

Σύμφωνα, δε, με το διεθνές πρότυπο ISO 8402, *ποιότητα είναι το σύνολο των χαρακτηριστικών γνωρισμάτων ενός προϊόντος ή μιας υπηρεσίας, που αφορούν τη δυνατότητα του προϊόντος ή της υπηρεσίας να ικανοποιήσει τις δηλωμένες ή υπονοούμενες ανάγκες.*

Ο όρος που χρησιμοποιείται σήμερα για την ποιότητα στις βιομηχανίες και τις υπηρεσίες ταυτίζεται με τη *Διοίκηση Ολικής Ποιότητας*. Παλιότερα οι επιχειρήσεις έψαχναν τρόπους να διορθώσουν τα προβλήματα ποιότητας εκ των υστέρων. Σήμερα σχεδιάζουν προληπτικούς τρόπους, ώστε να επιτύχουν την επιθυμητή ποιότητα κατά τη φάση του σχεδιασμού των προϊόντων και των υπηρεσιών.

2.2 Διοίκηση Ολικής Ποιότητας (TQM)

Η στρατηγική (φιλοσοφία) που επιζητεί τη συνεχή βελτίωση στην ποιότητα κατά την εκτέλεση όλων των διεργασιών παραγωγής προϊόντων και παροχής υπηρεσιών σε μια επιχείρηση ονομάζεται *Διοίκηση Ολικής Ποιότητας* (ΔΟΠ, TQM: Total Quality Management). Το σημαντικότερο χαρακτηριστικό γνώρισμα της ΔΟΠ (Διοίκησης Ολικής Ποιότητας) είναι η εστίαση της εταιρείας στους πελάτες της. Στόχος της είναι πρώτα να προσδιοριστούν οι ανάγκες των πελατών και στη συνέχεια να καλυφτούν. Η ΔΟΠ αναγνωρίζει ότι ένα απόλυτα τέλειο προϊόν έχει μικρή αξία, αν δεν είναι αυτό που θέλει ο πελάτης. Ως εκ τούτου, μπορούμε να πούμε ότι η ποιότητα εναρμονίζεται εν πολλοίς με τις απαιτήσεις του πελάτη. Δεν είναι όμως πάντοτε εύκολο να καθοριστεί τι ακριβώς θέλει ένας πελάτης, επειδή τα γούστα και οι προτιμήσεις μεταξύ των πελατών διαφέρουν.

Ένα άλλο χαρακτηριστικό γνώρισμα της φιλοσοφίας της ΔΟΠ είναι η έμφαση στη συνεχή βελτίωση (kaizen όπως ονομάζεται από τους Ιάπωνες (Brunet and New, 2003)).

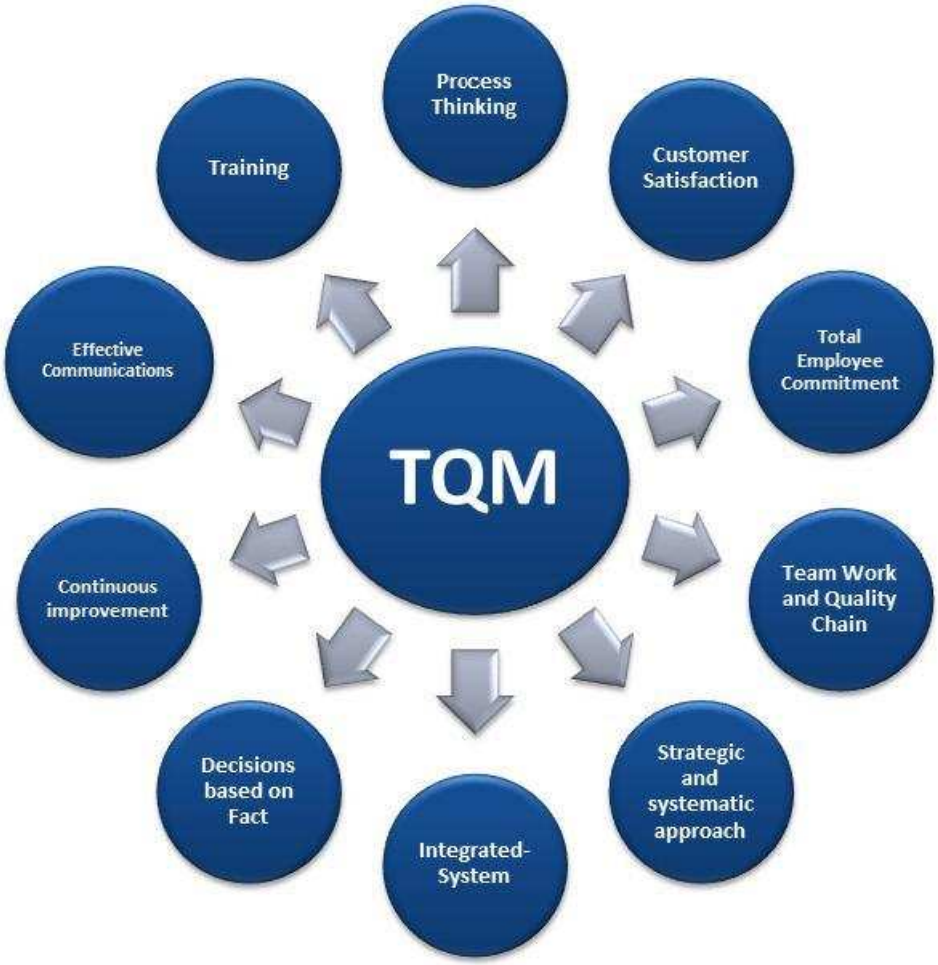
Τα παραδοσιακά συστήματα λειτουργούσαν με βάση την υπόθεση ότι μόλις μια εταιρεία επιτύχει ένα συγκεκριμένο επίπεδο ποιότητας και είναι επιτυχής τότε δε χρειάζονται περαιτέρω βελτιώσεις. Η ΔΟΠ απαιτεί από την εταιρεία να προσπαθεί συνεχώς να είναι καλύτερη μέσω της γνώσης και της επίλυσης προβλημάτων (βλ.Σχήμα 1). Στον παρακάτω πίνακα συνοψίζονται οι φιλοσοφίες των «γκουρού» της ποιότητας, οι οποίοι διαμόρφωσαν και εξέλιξαν τη Διοίκηση Ολικής Ποιότητας.

Walter A. Shewhart	
	- Συνέβαλε στην κατανόηση της μεταβλητότητας των διεργασιών - Ανέπτυξε τη θεωρία των Στατιστικών Διαγραμμάτων Ελέγχου
W. Edwards Deming	
	- Τόνισε την ευθύνη της διοίκησης για την ποιότητα - Ανέπτυξε τα 14 σημεία, ώστε να κατευθύνει τις επιχειρήσεις στη βελτίωση της ποιότητας
Joseph M. Juran	
	- Όρισε την ποιότητα ως «καταλληλότητα για χρήση» (fitness for use) - Ανέπτυξε την έννοια του κόστους ποιότητας
Armand V. Feigenbaum	
	- Εισήγαγε την έννοια του συνολικού ποιοτικού ελέγχου
Philip B. Crosby	
	- Επινόησε τη φράση «η ποιότητα είναι δωρεάν» - Εισήγαγε τη θεωρία των μηδενικών ελαττωμάτων
Kaoru Ishikawa	
	- Αναγνώρισε την έννοια του «εσωτερικού πελάτη» (internal customer) - Ανέπτυξε τα διαγράμματα Αιτίου-Αποτελέσματος (cause-and-effect).
Genichi Taguchi	
	- Εστίασε στην ποιότητα του σχεδιασμού των προϊόντων - Ανέπτυξε τη συνάρτηση απώλειας της ποιότητας

Πίνακας 1: *Οι «γκουρού» της Διοίκησης Ολικής Ποιότητας*

Σημαντικό ρόλο στη διαμόρφωση των νέων πλαισίων λειτουργίας και διοίκησης των επιχειρήσεων διαδραματίζει και ο *Διεθνής Οργανισμός Τυποποίησης* (ISO, International Standards Organization), ο οποίος με τα πρότυπά του καθορίζει τις διαδικασίες που

πρέπει να υιοθετηθούν από μια επιχείρηση, με τελικό σκοπό τη συνεχή προαγωγή και διασφάλιση της ποιότητας. Ενεργή και ιδιαίτερα σημαντική θέση στη ΔΟΠ κατέχει ο *Στατιστικός Έλεγχος Ποιότητας* (SQC: Statistical Quality Control).



Σχήμα 1: *Αρχές της Διοίκησης Ολικής Ποιότητας που καλύπτουν όλα τα πεδία διασφάλισης και διαχείρισης της ποιότητας σύμφωνα με τα πρότυπα ISO 9001, ISO 9004 και τις τεχνικές FMEA, KAIZEN, SPC, QA/QC*

3 Στατιστικός Έλεγχος Ποιότητας

Για την επίλυση των θεμάτων που μελετά η Διοίκηση Ολικής Ποιότητας απαιτούνται ειδικά εργαλεία που βοηθούν στη λήψη σωστών αποφάσεων. Μερικά από αυτά τα εργαλεία προέρχονται από την περιοχή της Στατιστικής και χρησιμοποιούνται για την αναγνώριση ποιοτικών προβλημάτων στις διεργασίες παραγωγής προϊόντων ή παροχής υπηρεσιών. Ο όρος που χρησιμοποιείται για να περιγράψει το σύνολο των στατιστικών εργαλείων που χρησιμοποιούνται από τους Μηχανικούς Ποιότητας ονομάζεται *Στατιστικός Έλεγχος Ποιότητας*.

Ο Στατιστικός Έλεγχος Ποιότητας αποτελεί την παλαιότερη και γνωστότερη μέθοδο ελέγχου διεργασιών για τη βελτίωση της ποιότητας των παραγόμενων προϊόντων (ή υπηρεσιών), προκειμένου να εξασφαλισθούν τα προκαθορισμένα ποιοτικά πρότυπα. Ο όρος χρησιμοποιείται για να περιγράψει το σύνολο των στατιστικών εργαλείων που χρησιμοποιούνται από τους επαγγελματίες της ποιότητας. Ένας από τους βασικούς στόχους του είναι η έγκαιρη ανακάλυψη μη συμμορφωμένων με τις προδιαγραφές παραγόμενων προϊόντων ή υπηρεσιών, η οποία σηματοδοτεί την ανάγκη λήψης διορθωτικών μέτρων για την απομάκρυνση των αιτιών που είναι υπεύθυνες για τις αποκλίσεις, συμβάλλοντας έτσι στη διατήρηση της ποιότητας.

3.1 Συνιστώσες του Στατιστικού Ελέγχου Ποιότητας

Ο Στατιστικός Έλεγχος Ποιότητας αποτελείται από ένα σύνολο μεθόδων στατιστικής ανάλυσης δεδομένων. Το σύνολο αυτό μπορεί να χωριστεί σε τρία βασικά υποσύνολα που το καθένα περιέχει στατιστικές μεθόδους προσανατολισμένες σε διαφορετικές φάσεις της παραγωγικής διαδικασίας, τα οποία είναι τα ακόλουθα:

- Σχεδιασμός και Ανάλυση Πειραμάτων (Design of Experiments)
- Στατιστικός Έλεγχος Διεργασιών (Statistical Process Control)
- Δειγματοληψία Αποδοχής (Acceptance Sampling)

Ο *Σχεδιασμός και η Ανάλυση Πειραμάτων* περιέχει όλες εκείνες τις στατιστικές τεχνικές οι οποίες μας βοηθούν στην ανακάλυψη της επίδρασης που έχουν τα διάφορα επίπεδα των παραγόντων (μεταβλητών) που επηρεάζουν τις ποιοτικές παραμέτρους του τελικού προϊόντος και συνεπώς διαδραματίζει σημαντικό ρόλο στη βέλτιστη σχεδίαση της παραγωγικής διεργασίας. Ο *Στατιστικός Έλεγχος Διεργασιών* περιέχει στατιστικές τεχνικές που είναι απαραίτητες για τον έλεγχο της παραγωγικής διεργασίας κατά την διάρκεια της παραγωγής των προϊόντων. Η *Δειγματοληψία Αποδοχής* περιέχει στατιστικές τεχνικές (δειγματοληπτικές) που είναι απαραίτητες για να αποφασιστεί αν μια συγκεκριμένη παρτίδα προϊόντων μπορεί να γίνει δεκτή ή αν θα απορριφθεί. Στο επόμενο κεφάλαιο εξετάζεται πιο αναλυτικά ο Στατιστικός Έλεγχος Διεργασιών, ο οποίος περιλαμβάνει μέσα στις τεχνικές του και τα Διαγράμματα Ελέγχου.

3.2 Στατιστικός Έλεγχος Διεργασιών

Η επιστήμη της Στατιστικής έχει προσφέρει ορισμένα πολύ σημαντικά εργαλεία για την αποτελεσματική λειτουργία όλων των σύγχρονων Συστημάτων Διαχείρισης της Ποιότητας (SQM: System Quality Management). Ουσιαστικά τα εργαλεία αυτά παρέχουν τη δυνατότητα στο σύστημα ποιοτικού ελέγχου να διακρίνει εύκολα και με μικρό κόστος τα προϊόντα που δε συμμορφώνονται με τις προκαθορισμένες προδιαγραφές. Σε αυτές τις περιπτώσεις ενεργοποιούν τις ενδεδειγμένες διαδικασίες διορθωτικών ενεργειών, διατηρώντας έτσι την ποιότητα σε ανταγωνιστικό επίπεδο.

Ο Στατιστικός Έλεγχος Διεργασιών (ΣΕΔ, SPC: Statistical Process Control) είναι μια από τις τεχνικές της Ολικής Διοίκησης Ποιότητας (Total Quality Management) που χρησιμοποιείται στη βελτίωση της ποιότητας του προϊόντος, της παραγωγικής διαδικασίας και των υπηρεσιών. Ο ΣΕΔ στοχεύει στην πρόληψη, τη συνεχή βελτίωση και την ενσωμάτωση της ποιότητας σε όλα τα στάδια της παραγωγικής διεργασίας. Ένας ορισμός που έχει δοθεί για τον ΣΕΔ (Montgomery, 1997) είναι ο εξής:

«Ο Στατιστικός Έλεγχος Διεργασιών είναι ένα σύνολο ισχυρών εργαλείων

επίλυσης προβλημάτων χρήσιμων για την επίτευξη σταθερής διεργασίας και τη βελτίωση της ικανότητάς της μέσω της μείωσης της μεταβλητότητας.»

Ο Στατιστικός Έλεγχος Διεργασιών, λοιπόν, είναι μία μέθοδος παρακολούθησης, ελέγχου, ανάλυσης και βελτίωσης μιας διεργασίας με στόχο την πρόληψη και τη συστηματική μείωση της μεταβλητότητας. Χρησιμοποιεί πολλά εργαλεία για την επίτευξη αυτού του στόχου, από τα οποία το πρώτο αναπτύχθηκε από τον W.A.Shewhart και είναι τα λεγόμενα *διαγράμματα ελέγχου ή διαγράμματα Shewhart*, τα οποία περιγράφονται στο Κεφάλαιο 4.

Ο ΣΕΔ χρησιμοποιείται ευρέως στη βιομηχανία όπου οι διαδικασίες παραγωγής πρέπει να βρίσκονται σε διαρκή έλεγχο. Η ανάγκη παρακολούθησης συγκεκριμένων διαδικασιών έχει οδηγήσει στη μεγάλη ανάπτυξη και βελτίωση αυτών των μεθόδων. Τα εργαλεία του ΣΕΔ που έχουν αναπτυχθεί χρησιμοποιούνται και σε διάφορους τομείς της επιστήμης, όπως στην ιατρική, το περιβάλλον, την οικονομία, την ανάλυση κειμένου καθώς και στην πληροφορική.

3.3 Μεταβλητότητα Διεργασιών

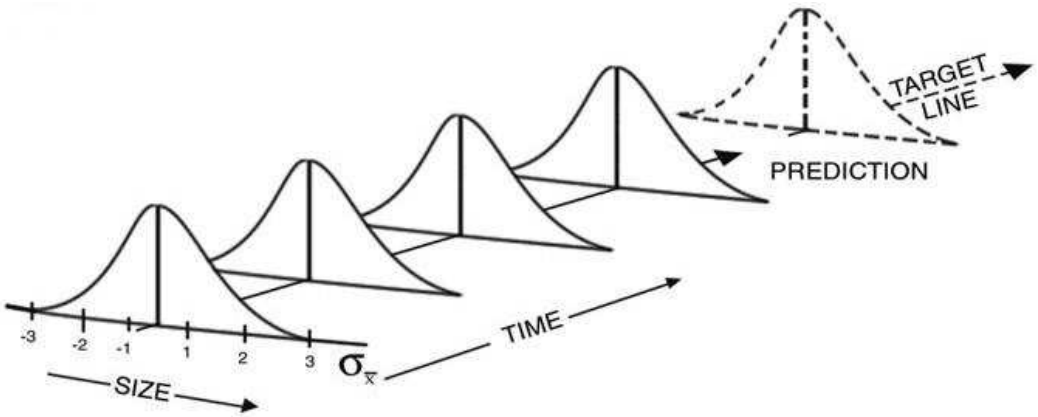
Προς το τέλος του 1920 ο Dr. Walter A. Shewhart αρχικά και αργότερα ο Dr. W. Edwards Deming παρατήρησαν ότι σε κάθε διεργασία υπάρχει μεταβλητότητα (variability). Η μεταβλητότητα αυτή μπορεί να οφείλεται είτε σε *τυχαία αίτια* είτε σε *ειδικά αίτια*. Η εξασφάλιση και η διατήρηση της ποιότητας των προϊόντων και υπηρεσιών απαιτεί την κατανόηση, την παρακολούθηση και τον έλεγχο της μεταβλητότητας με στόχο τη μείωσή της στο ελάχιστο.

Σε κάθε παραγωγική διεργασία, ανεξάρτητα από το πόσο καλά σχεδιασμένη είναι και το πόσο προσεκτικά επιβλέπεται και συντηρείται, θα υπάρχει πάντα μια μορφή φυσικής μεταβλητότητας που θα τη συνοδεύει. Δηλαδή, όσο καλά ρυθμισμένα και να είναι τα μηχανήματα, όσο ικανοί και να είναι οι χειριστές των μηχανημάτων, όσο ικανοποιητική και να είναι η πρώτη ύλη, ποτέ δύο παραγόμενα προϊόντα δεν θα είναι τα ίδια. Θα υπάρχει

κάποιο μετρήσιμο μέγεθος του προϊόντος του οποίου η τιμή θα είναι διαφορετική ανάμεσα στα προϊόντα.

Αυτή η φυσική μεταβλητότητα είναι το αθροιστικό αποτέλεσμα πολλών μικρών αιτιών, οι οποίες αναφέρονται ως *τυχαίες αιτίες μεταβλητότητας* (random causes). Η φυσική μεταβλητότητα είναι συνήθως μικρή σε μέγεθος και δεν μπορεί να αποδοθεί σε ελέγξιμους παράγοντες. Οι τυχαίες αιτίες της μεταβλητότητας είναι ενσωματωμένες στη διεργασία και αποτελούν αναπόσπαστο κομμάτι της. Ως συνέπεια αυτού η πλήρης εξάλειψη των τυχαίων αιτιών δεν είναι εφικτή.

Οι διεργασίες που βρίσκονται κάτω από την επίδραση μόνο τυχαίων αιτιών θεωρούνται *σταθερές*, *προβλέψιμες* και *υπό στατιστικό έλεγχο* (in statistical control process) και λέμε ότι λειτουργούν σε *ευσταθή κατάσταση* (stable state, Σχήμα 2). Η σωστή ρύθμιση των παραμέτρων τέτοιων διεργασιών έχει σαν αποτέλεσμα τη συστηματική παραγωγή προϊόντων και υπηρεσιών με την απαιτούμενη ποιότητα και εντός των προκαθορισμένων προδιαγραφών.



Σχήμα 2: *Διεργασία εντός στατιστικού ελέγχου*

Σε μια διεργασία, όμως, μπορεί να εμφανίζονται περιστασιακά και άλλες μορφές μεταβλητότητας, οι οποίες δεν οφείλονται σε τυχαίες αιτίες, αλλά αφορούν τη συστηματική αλλαγή στο επίπεδο κάποιων παραγόντων που καθορίζουν την ποιότητα του

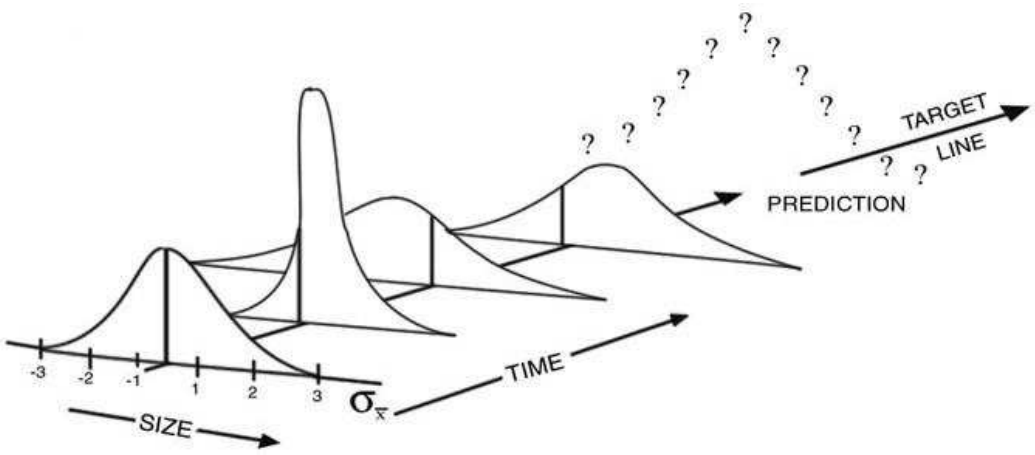
προϊόντος. Αυτές οι μορφές μεταβλητότητας οφείλονται συνήθως στους ακόλουθους λόγους:

(α) λανθασμένα ρυθμισμένες μηχανές,

(β) λάθη των χειριστών των μηχανημάτων, και

(γ) κακής ποιότητας ή ελαττωματική πρώτη ύλη.

Η μεταβλητότητα που οφείλεται στους παραπάνω λόγους είναι σε μέγεθος πολύ μεγαλύτερη της φυσικής μεταβλητότητας και η παρουσία της οδηγεί συνήθως σε μη αποδεκτά επίπεδα λειτουργίας της παραγωγικής διεργασίας. Αυτή η μεταβλητότητα αναφέρεται ως *ειδική μεταβλητότητα* και οι αιτίες που οδηγούν σε αυτή ονομάζονται *ειδικές αιτίες μεταβλητότητας* (special causes of variation). Μια διεργασία που λειτουργεί με την παρουσία ειδικής μεταβλητότητας λέμε ότι είναι *ασταθής*, *απρόβλεπτη* και *εκτός στατιστικού ελέγχου* (out of statistical control process) ή ότι λειτουργεί σε *ασταθή κατάσταση* (unstable state, Σχήμα 3).



Σχήμα 3: *Διεργασία εκτός στατιστικού ελέγχου*

Ο στατιστικός έλεγχος διεργασιών αποσκοπεί στο διαχωρισμό των τυχαίων αιτιών μεταβλητότητας από τις ειδικές αιτίες, οι οποίες πρέπει να εξαλειφθούν μετά από τον εντοπισμό τους. Η διάκριση και η κατανόηση της διαφοράς μεταξύ αυτών των δύο

αιτιών μεταβλητότητας είναι ιδιαίτερα σημαντική για την αποτελεσματικότερη βελτίωση της ποιότητας. Για τη διάκριση των δύο αιτιών μεταβλητότητας ο Dr. W.A.Shewhart επινόησε τα διαγράμματα ελέγχου ένα από τα πιο παλιά εργαλεία του ΣΕΔ, τα οποία μελετούνται πιο αναλυτικά στο Κεφάλαιο 4.

3.4 Βασικές Στατιστικές Έννοιες

Στις επόμενες παραγράφους γίνεται μια σύντομη αναφορά σε κάποιες βασικές έννοιες της Στατιστικής, όπως είναι η διακύμανση και η τυπική απόκλιση. Επίσης, περιγράφεται η Κανονική Κατανομή και το Κεντρικό Οριακό Θεώρημα, το οποίο αποτελεί και την θεωρητική βάση των διαγραμμάτων ελέγχου μεταβλητών.

3.4.1 Διακύμανση και Τυπική Απόκλιση

Η διακύμανση ή η διασπορά (variance) μιας τυχαίας μεταβλητής x είναι ο μέσος όρος των τετραγώνων των αποκλίσεων των διαφόρων μετρήσεων x_i της μεταβλητής x από τη μέση τιμή τους μ. Ενώ η τυπική απόκλιση (standard deviation) είναι η τετραγωνική ρίζα της διακύμανσης.

Η διακύμανση είναι μια αξιόπιστη παράμετρος διασποράς, αλλά δεν χρησιμοποιείται τόσο όταν πρόκειται να μελετηθεί η συμπεριφορά της μεταβλητότητας μιας διεργασίας. Αυτό οφείλεται στο γεγονός ότι το μέτρο της διακύμανσης δεν εκφράζεται με τις ίδιες μονάδες με τις οποίες εκφράζονται οι παρατηρήσεις x_i. Έτσι, για τη μελέτη της μεταβλητότητας μιας διεργασίας χρησιμοποιείται η τυπική απόκλιση, αφού είναι ένα μέτρο διασποράς που εκφράζεται με την ίδια μονάδα μέτρησης με την οποία εκφράζεται και η μεταβλητή x.

Ο στατιστικός τύπος τόσο για τη διακύμανση όσο και για την τυπική απόκλιση διαφέρει ανάλογα με το αν μιλάμε για όλο τον πληθυσμό μιας αριθμητικής σειράς ή για ένα μέρος της, ένα δείγμα της. Η αλλαγή είναι μικρή, αλλά πολλές φορές σημαντική όταν πρόκειται για ένα στατιστικό έλεγχο. Έτσι, όταν μιλάμε για όλο τον πληθυσμό N μίας

αριθμητικής σειράς με μέση τιμή μ, για τον υπολογισμό της διακύμανσης σ^2 και της τυπικής απόκλισης σ χρησιμοποιούνται οι παρακάτω τύποι:

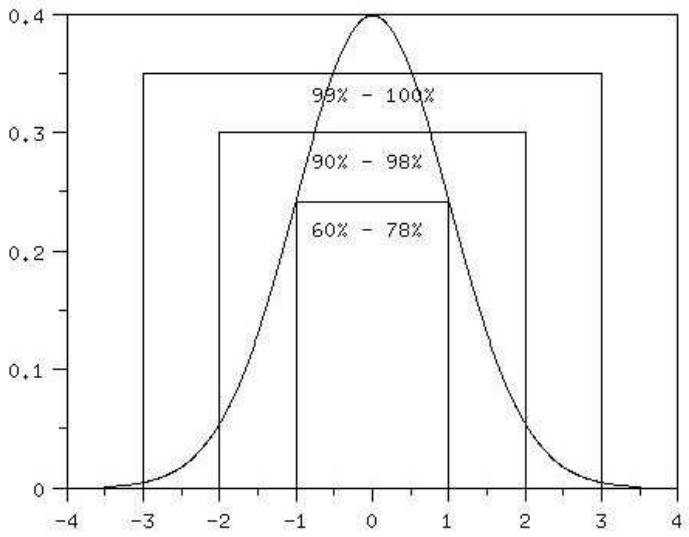
$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$
$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3.4.1}$$

ενώ όταν μιλάμε για ένα δείγμα μόνο της ίδιας αριθμητικής σειράς τότε η διακύμανση συμβολίζεται ως S^2 ενώ η τυπική απόκλιση ως $\sqrt{S^2}$ και υπολογίζονται από τους παρακάτω τύπους:

$$S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$$
$$\sqrt{S^2} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3.4.2}$$

Η τυπική απόκλιση σαν μέγεθος μας δίνει μια εικόνα για το πώς εξαπλώνονται τα δεδομένα σε σχέση πάντα με τη μέση τιμή. Στην Στατιστική χρησιμοποιείται ένας κανόνας, γνωστός ως *Εμπειρικός Κανόνας*, ο οποίος ονομάστηκε έτσι επειδή επαληθεύεται συχνά εμπειρικά σε διάφορα πειράματα και φαινόμενα. Σύμφωνα με τον Εμπειρικό Κανόνα:

- Περίπου το $60-78\%$ των δεδομένων βρίσκεται σε απόσταση μίας τυπικής απόκλισης από τη μέση τιμή $(\mu - \sigma, \mu + \sigma)$.
- Περίπου το $90 - 98\%$ των δεδομένων βρίσκεται σε απόσταση δύο τυπικών αποκλίσεων από τη μέση τιμή $(\mu - 2\sigma, \mu + 2\sigma)$.
- Παραπάνω από το 99% των δεδομένων βρίσκεται σε απόσταση τριών τυπικών αποκλίσεων από τη μέση τιμή $(\mu - 3\sigma, \mu + 3\sigma)$.



Σχήμα 4: *Εμπειρικός Κανόνας*

3.4.2 Η Κανονική Κατανομή

Η *Κανονική Κατανομή* (normal distribution) ανακαλύφθηκε γύρω στο 1720 από τον Μαθηματικό Abraham De Moivre στην προσπάθειά του να λύσει προβλήματα παιγνίων τύχης. Εκατόν πενήντα χρόνια αργότερα περίπου το 1870, ο Βέλγος Μαθηματικός Adolph Quetelet άρχισε να χρησιμοποιεί την καμπύλη της κανονικής κατανομής ως το *ιδεώδες οριακό ιστόγραμμα* (πρότυπο), προς το οποίο συγκρίνονται τα ιστογράμματα δεδομένων.

Πλέον, στις μέρες μας η κανονική κατανομή θεωρείται η σπουδαιότερη κατανομή της Θεωρίας Πιθανοτήτων και της Στατιστικής. Οι λόγοι που εξηγούν την εξέχουσα θέση της είναι οι εξής:

i) Πολλά πειράματα μπορούν να εκφρασθούν μέσω τυχαίων μεταβλητών που ακολουθούν την κανονική κατανομή.

ii) Η κανονική κατανομή μπορεί να χρησιμοποιηθεί σαν προσέγγιση πολλών άλλων κατανομών (Κεντρικό Οριακό Θεώρημα).

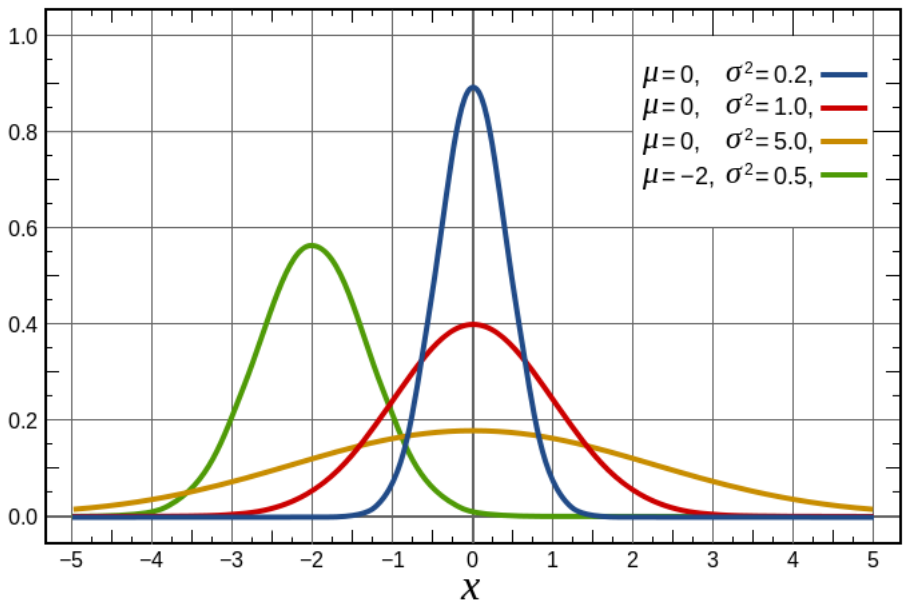
iii) Η κατανομή αυτή αποτελεί την βάση για πολλές τεχνικές που χρησιμοποιούνται στην στατιστική συμπερασματολογία.

Ορισμός 3.1. Έστω x μια συνεχής τυχαία μεταβλητή. Λέμε ότι η μεταβλητή x ακολουθεί την κανονική κατανομή (normal distribution) με παραμέτρους μ και σ^2 με $\sigma > 0$, το οποίο συμβολίζεται ως $x \sim N(\mu, \sigma^2)$, αν ισχύει:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{3.4.3}$$

όπου $-\infty < \mu < +\infty$ και $-\infty < x < +\infty$. Η παραπάνω σχέση ορίζει την $f(x)$, η οποία είναι η συνάρτηση πυκνότητας πιθανότητας της τυχαίας μεταβλητής x.

□



Σχήμα 5: *Κανονική Κατανομή*

Η κανονική κατανομή παρουσιάζει τις εξής ιδιότητες:

α) Είναι συμμετρική γύρω από το σημείο $x = \mu$.

β) Έχει σχήμα καμπάνας με επικρατούσα τιμή στο σημείο $x = \mu$, όπως φαίνεται και στο σχήμα 5. Στο σημείο αυτό $f(x) = 1/(\sigma\sqrt{2\pi})$ και επομένως το σ προσδιορίζει το μέγιστο της συνάρτησης. Είναι φανερό ότι η συνάρτηση $f(x)$ στο σημείο $x = \mu$ είναι αντιστρόφως ανάλογη της τιμής του σ.

3.4.3 Κεντρικό Οριακό Θεώρημα

Ο *νόμος των μεγάλων αριθμών* προσδιορίζει τη μορφή σύγκλισης ακολουθιών τυχαίων μεταβλητών, οι οποίες μπορούν να εκφρασθούν ως μερικά αθροίσματα άλλων ανεξαρτήτων τυχαίων μεταβλητών. Συγκεκριμένα, λέει, ότι αν έχουμε μια τυχαία μεταβλητή της μορφής,

$$S_n = X_1 + X_2 + ... + X_n$$

τότε αυτή συγκλίνει στη μέση της τιμή, όπως αυτή ορίζεται από τη σχέση:

$$E(S_n) = \sum_{i=1}^{n} E(X_i)$$

Το επόμενο ερώτημα που προκύπτει για την τυχαία αυτή μεταβλητή S_n είναι το ποιά κατανομή θα ακολουθεί. Την απάντηση στο ερώτημα αυτό μας τη δίνει το *Κεντρικό Οριακό Θεώρημα*, το οποίο αποδεικνύει ότι η κατανομή της τυχαίας μεταβλητής S_n είναι κατά προσέγγιση η κανονική κατανομή. Μία στατιστική διατύπωση του Κεντρικού Οριακού Θεωρήματος το ορίζει ως εξής:

Ορισμός 3.2. Έστω μια τυχαία μεταβλητή X, η οποία περιγράφει τον υπό μελέτη πληθυσμό και ακολουθεί μια τυχαία κατανομή με πεπερασμένη μέση τιμή μ και πεπερασμένη διασπορά σ_X^2. Αν από την μεταβλητή αυτή λάβουμε n αμοιβαία ανεξάρτητες παρατηρήσεις $X_1, X_2, ..., X_n$ τότε ο μέσος όρος των παρατηρήσεων αυτών, ο οποίος καλείται *δειγματικός μέσος* υπολογίζεται από την παρακάτω σχέση:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Σύμφωνα με το Κεντρικό Οριακό Θεώρημα, για μεγάλες τιμές του n (θεωρητικά $n \to \infty$) η κατανομή αυτών των δειγματικών μέσων είναι κατά προσέγγιση μια κανονική κατανομή με μέση τιμή επίσης μ και διασπορά σ_X^2/n. Δηλαδή,

$$\bar{\bar{X}} = \mu, \quad \sigma_{\bar{X}}^2 = \frac{\sigma_X^2}{n}, \quad \bar{X} \sim N(\mu, \sigma^2/n)$$

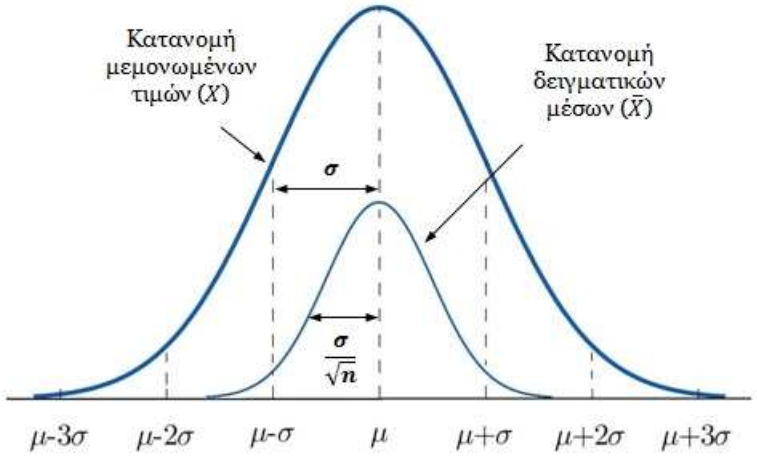
Επίσης, αν θεωρήσουμε τη μεταβλητή:

$$Z_n = \frac{\bar{X} - \mu}{\sigma\sqrt{n}}$$

τότε η μεταβλητή αυτή θα ακολουθεί την τυποποιημένη κανονική κατανομή, δηλαδή

$$Z_n \sim N(0, 1)$$

□



Σχήμα 6: *Κεντρικό Οριακό Θεώρημα*

Με βάση το θεώρημα του κεντρικού ορίου οι ιδιότητες της κανονικής κατανομής ισχύουν και για τον πληθυσμό των μέσων του δείγματος. Συγκεκριμένα, μπορούμε να πούμε ότι το 99.7% των μέσων του δείγματος κατανέμονται μέσα σε μια ζώνη ± 3 τυπικών

αποκλίσεων ($\pm 3\sigma$) μετρημένες από την κεντρική τιμή (μέση τιμή). Αυτές οι γραμμές των ± 3 τυπικών αποκλίσεων ορίζουν τα όρια ελέγχου στο διάγραμμα ελέγχου. Η πιθανότητα να πέσει ένα σημείο (ένας μέσος δείγματος) εκτός των ορίων ελέγχου είναι της τάξης του 0.0027 ή αλλιώς 3% κατά προσέγγιση. Η πιθανότητα αυτή είναι τόσο μικρή που εάν συμβεί μπορούμε να θεωρήσουμε ότι δεν συνέβη τυχαία, αλλά λόγω της παρουσίας ενός ειδικού αιτίου διασποράς.

3.5 Πολυμεταβλητός Στατιστικός Έλεγχος Ποιότητας

Η παρακολούθηση κάθε διεργασίας, η ανίχνευση σφαλμάτων και η διάγνωσή τους σε αρχικό στάδιο είναι ιδιαίτερα σημαντικές για τη διασφάλιση της ποιότητας σε όλους τους οργανισμούς. Ο *Πολυμεταβλητός Στατιστικός Έλεγχος Ποιότητας* δίνει τη δυνατότητα ταυτόχρονης παρακολούθησης δύο ή περισσοτέρων μεταβλητών ενός προϊόντος. Η ανάγκη χρησιμοποίησης του Πολυμεταβλητού Στατιστικού Ελέγχου Ποιότητας προέκυψε από τη διαπίστωση ότι η ποιότητα ενός προϊόντος μπορεί να σχετίζεται με περισσότερα του ενός ποιοτικά και μετρήσιμα χαρακτηριστικά.

Οι περισσότερες διεργασίες παραγωγής προϊόντων ή παροχής υπηρεσιών είναι πολυμεταβλητές, εξαρτώνται δηλαδή από περισσότερες από μια μεταβλητές. Αν και μια πιθανή λύση για την παρακολούθηση της ποιότητας μιας τέτοιας διεργασίας είναι η εφαρμογή των Διαγραμμάτων Ελέγχου σε κάθε μεταβλητή χωριστά, αυτό κρίνεται ανεπαρκές και μπορεί να οδηγήσει σε λανθασμένα συμπεράσματα. Γί αυτό απαιτούνται πολυμεταβλητές μέθοδοι που εξετάζουν τις μεταβλητές από κοινού.

Λόγω του γεγονότος ότι οι υπολογισμοί κατά τον πολυμεταβλητό στατιστικό έλεγχο είναι αρκετά περίπλοκοι και επίπονοι, καθώς απαιτούν γνώση της Θεωρίας των Πινάκων, η αποδοχή των πολυμεταβλητών διαγραμμάτων ελέγχου από τη βιομηχανία ήταν αργή και διστακτική. Στις μέρες μας, πλέον, με τους σύγχρονους υπολογιστές τέτοιοι πολύπλοκοι υπολογισμοί γίνονται εφικτοί σε απίστευτες ταχύτητες. Στη συνέχεια, παρουσιάζονται κάποιες από τις βασικές έννοιες της πολυμεταβλητής ανάλυσης, οι οποίες χρησιμοποιού-

νται για την κατασκευή πολυμεταβλητών διαγραμμάτων ελέγχου.

3.5.1 Πίνακας Συνδιακύμανσης και Διάνυσμα Μέσων

Στη θεωρία της στατιστικής και των πιθανοτήτων η συνδιακύμανση είναι ένα μέτρο που μας δείχνει το πόσο δύο τυχαίες μεταβλητές αλλάζουν μαζί. Έστω ένα διάνυσμα $\mathbf{X}$ το οποίο αποτελείται από n τυχαίες μεταβλητές:

$$\mathbf{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \tag{3.5.1}$$

τότε η συνδιακύμανση Σ δύο τυχαίων μεταβλητών X_i και X_j ορίζεται ως εξής:

$$\Sigma_{ij} = cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] \tag{3.5.2}$$

όπου μ_i, είναι η μέση τιμή της τυχαίας μεταβλητής X_i και αντίστοιχα η μ_j είναι η μέση τιμή της τυχαίας μεταβλητής X_j δηλαδή,

$$\begin{aligned} \mu_i &= E(X_i) \\ \mu_j &= E(X_j) \end{aligned} \tag{3.5.3}$$

Η συνδιακύμανση μπορεί να υπολογιστεί και από τον τύπο:

$$cov(X_i, X_j) = \frac{\sum_{i=1}^{n}(X_i - \mu_i)(X_j - \mu_j)}{(n-1)} \tag{3.5.4}$$

Ο πίνακας συνδιακύμανσης εκφράζεται από την παρακάτω σχέση:

$$\Sigma = \begin{bmatrix} E[(X_1-\mu_1)(X_1-\mu_1)] & E[(X_1-\mu_1)(X_2-\mu_2)] & \dots & E[(X_1-\mu_1)(X_n-\mu_n)] \\ E[(X_2-\mu_2)(X_1-\mu_1)] & E[(X_2-\mu_2)(X_2-\mu_2)] & \dots & E[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(X_n-\mu_n)(X_1-\mu_1)] & E[(X_n-\mu_n)(X_2-\mu_2)] & \dots & E[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix} \tag{3.5.5}$$

όπου $E[(X_i-\mu_i)(X_i-\mu_i)]$ είναι η διακύμανση της μεταβλητής X_i, δηλαδή,

$$var(X_i) = \frac{\sum_{i=1}^{n}(X_i-\mu_i)^2}{(n-1)} \tag{3.5.6}$$

Σε πιο απλοποιημένη μορφή ο πίνακας συνδιακύμανσης μπορεί να γραφεί ως εξής:

$$\Sigma = \begin{bmatrix} var(X_1) & cov(X_1,X_2) & \dots & cov(X_1,X_n) \\ cov(X_2,X_1) & var(X_2) & \dots & cov(X_2,X_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(X_n,X_1) & cov(X_n,X_2) & \dots & var(X_n) \end{bmatrix} \tag{3.5.7}$$

Παράδειγμα 3.1. Για να γίνουν πιο κατανοητές οι παραπάνω σχέσεις θα εφαρμόσουμε ένα παράδειγμα τριών μεταβλητών. Έστω ότι οι μεταβλητές αυτές είναι το μήκος, το πλάτος και το ύψος ενός αντικειμένου και έστω ότι παίρνουμε 5 μετρήσεις από την κάθε μία. Τα δεδομένα μας μπορούμε να τα εκφράσουμε σε μορφή πίνακα και ας θεωρήσουμε ότι είναι τα εξής:

$$x = \begin{bmatrix} 4.0 & 2.0 & 0.60 \\ 4.2 & 2.1 & 0.59 \\ 3.9 & 2.0 & 0.58 \\ 4.3 & 2.1 & 0.62 \\ 4.1 & 2.2 & 0.63 \end{bmatrix}$$

στον οποίο η κάθε στήλη περιέχει τις 5 μετρήσεις της αντίστοιχης μεταβλητής (X_1:

μήκος, X_2: πλάτος, X_3: ύψος). Στη συνέχεια, ορίζουμε το *διάνυσμα των μέσων*, το οποίο αποτελείται ουσιαστικά από τις μέσες τιμές των τριών μεταβλητών. Δηλαδή:

$$\bar{\mathbf{x}} = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & \bar{x}_3 \end{bmatrix} \tag{3.5.8}$$

όπου σύμφωνα με τον τύπο για τη μέση τιμή θα έχουμε,

$$\begin{aligned} \bar{x}_1 &= \frac{4.0 + 4.2 + 3.9 + 4.3 + 4.1}{5} = 4.10 \\ \bar{x}_1 &= \frac{2.0 + 2.1 + 2.0 + 2.1 + 2.2}{5} = 2.08 \\ \bar{x}_1 &= \frac{0.60 + 0.59 + 0.58 + 0.62 + 0.63}{5} = 0.604 \end{aligned}$$

και επομένως το διάνυσμα των μέσων θα γράφεται ως εξής:

$$\bar{\mathbf{x}} = \begin{bmatrix} 4.10 & 2.08 & 0.604 \end{bmatrix}$$

Ο πίνακας συνδιακύμανσης σύμφωνα με τη σχέση 3.5.7, διαμορφώνεται ως εξής:

$$\mathbf{\Sigma} = \begin{bmatrix} var(x_1) & cov(x_1, x_2) & cov(x_1, x_3) \\ cov(x_2, x_1) & var(x_2) & cov(x_2, x_3) \\ cov(x_3, x_1) & cov(x_3, x_2) & var(x_3) \end{bmatrix}$$

Οι διασπορές $var(x_i)$ της κάθε μεταβλητής σύμφωνα με τη σχέση 3.5.6, παίρνουν την παρακάτω μορφή,

$$\begin{aligned} var(x_1) &= \left[(4.0 - 4.1)^2 + (4.2 - 4.1)^2 + (3.9 - 4.1)^2 + (4.3 - 4.1)^2 + (4.1 - 4.1)^2\right]/4 \\ var(x_2) &= \left[(2.0 - 2.08)^2 + (2.1 - 2.08)^2 + (2.0 - 2.08)^2 + (2.1 - 2.08)^2 + (2.2 - 2.08)^2\right]/4 \\ var(x_3) &= \left[(0.6 - 0.604)^2 + (0.59 - 0.604)^2 + (0.58 - 0.604)^2 + (0.62 - 0.604)^2 + (0.63 - 0.604)^2\right]/4 \end{aligned}$$

όπου μετά από πράξεις προκύπτουν τα εξής αποτελέσματα:

$$var(x_1) = 0.025$$
$$var(x_2) = 0.007$$
$$var(x_3) = 0.00043$$

Τέλος, για τις συνδιακυμάνσεις $cov(x_i, x_j)$ χρησιμοποιώντας τον τύπο 3.5.4 θα έχουμε,

$$\begin{aligned} cov(x_1, x_2) &= [(4.0 - 4.10)(2.0 - 2.08) + (4.2 - 4.10)(2.1 - 2.08) + (3.9 - 4.10)(2.0 - 2.08) + \\ &+ (4.3 - 4.10)(2.1 - 2.08) + (4.1 - 4.10)(2.2 - 2.08)]/4 \\ cov(x_2, x_3) &= [(2.0 - 2.08)(0.6 - 0.604) + (2.1 - 2.08)(0.59 - 0.604) + (2.0 - 2.08)(0.58 - 0.604) + \\ &+ (2.1 - 2.08)(0.62 - 0.604) + (2.2 - 2.08)(0.63 - 0.604)]/4 \\ cov(x_1, x_3) &= [(4.0 - 4.10)(0.6 - 0.604) + (4.2 - 4.10)(0.59 - 0.604) + (3.9 - 4.10)(0.58 - 0.604) + \\ &+ (4.3 - 4.10)(0.62 - 0.604) + (4.1 - 4.10)(0.63 - 0.604)]/4 \end{aligned}$$

όπου μετά από πράξεις προκύπτουν τα εξής αποτελέσματα:

$$cov(x_1, x_2) = cov(x_2, x_1) = 0.0075$$
$$cov(x_2, x_3) = cov(x_3, x_2) = 0.00135$$
$$cov(x_1, x_3) = cov(x_1, x_3) = 0.00175$$

Και η τελική μορφή του πίνακα συνδιακύμανσης είναι η ακόλουθη,

$$\mathbf{\Sigma} = \begin{bmatrix} 0.025 & 0.0075 & 0.00175 \\ 0.0075 & 0.0070 & 0.00135 \\ 0.00175 & 0.00135 & 0.00043 \end{bmatrix}$$

♦

3.5.2 Πολυμεταβλητή Κανονική Κατανομή

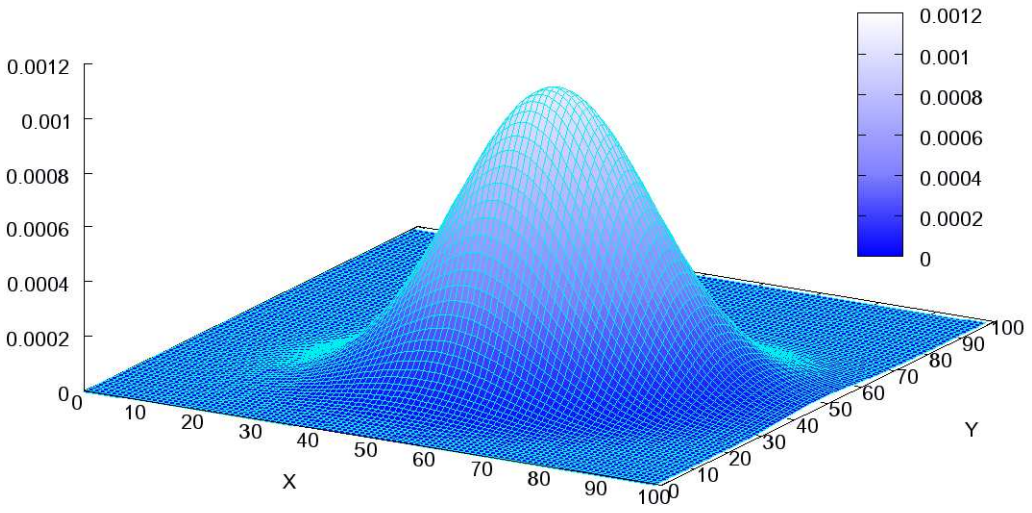
Στη Θεωρία των Πιθανοτήτων και της Στατιστικής η πολυμεταβλητή Κανονική Κατανομή είναι η γενίκευση της μονοδιάστατης Κανονικής Κατανομής σε πολλές διαστάσεις (μεταβλητές) και μπορεί να χρησιμοποιηθεί στις περιπτώσεις που μελετούνται πολυμεταβλητά προβλήματα ή προβλήματα με διανύσματα.

Ορισμός 3.3. Έστω ένα διάνυσμα p-συντεταγμένων, $\mathbf{x} = [x_1, x_2, ..., x_p]$, για το οποίο ισχύει ότι $-\infty < x_i < +\infty$ με $i = 1, 2, ..., p$. Λέμε ότι το διάνυσμα $\mathbf{x}$ ακολουθεί την πολυδιάστατη Κανονική Κατανομή, το οποίο συμβολίζεται ως $\mathbf{x} \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, αν η συνάρτηση πυκνότητας πιθανότητας του $\mathbf{x}$, $f(\mathbf{x})$ έχει την εξής μορφή:

$$f(\mathbf{x}) = f(x_1, x_2, ..., x_p) = \left(\frac{1}{2\pi}\right)^{p/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right] \quad (3.5.9)$$

όπου $\boldsymbol{\mu}$ είναι το διάνυσμα των μέσων και $\boldsymbol{\Sigma}$ είναι ο πίνακας συνδιακύμανσης της πολυδιάστατης κανονικής κατανομής.

□



Σχήμα 7: *Συνάρτηση Πυκνότητας Πιθανότητας Δισδιάστατης Κανονικής Κατανομής*

4 Διαγράμματα Ελέγχου Shewhart

Τα διαγράμματα ελέγχου επινοήθηκαν από τον Walter A. Shewhart, ενώ εργαζόταν στην Bell Labs το 1920. Οι μηχανικοί της εταιρείας επεδίωκαν να βελτιώσουν την αξιοπιστία των συστημάτων μετάδοσης της τηλεφωνίας τους. Επειδή οι ενισχυτές και ο υπόλοιπος εξοπλισμός έπρεπε να τοποθετηθεί κάτω από την επιφάνεια του εδάφους, υπήρξε μια ισχυρότερη ανάγκη της επιχείρησης για να μειωθεί η συχνότητα των βλαβών και των επισκευών.

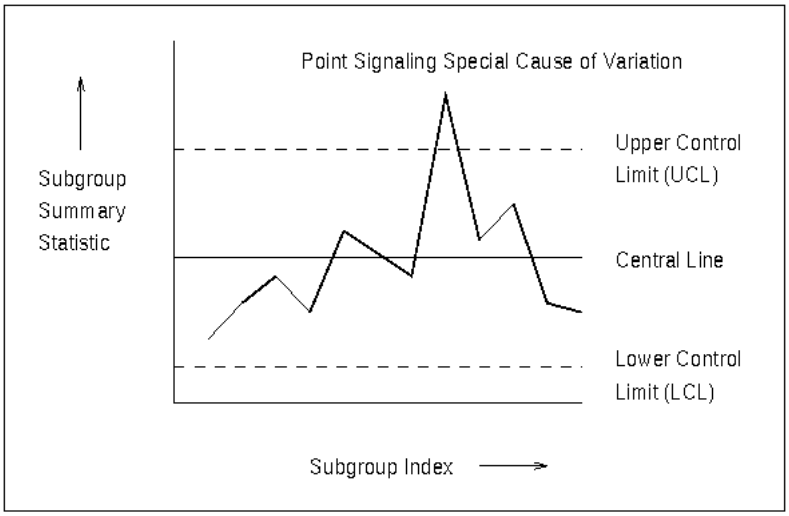
Μέχρι το 1920, οι μηχανικοί είχαν ήδη συνειδητοποιήσει τη σημασία της μείωσης της διακύμανσης σε μια διαδικασία κατασκευής. Επιπλέον, είχαν συνειδητοποιήσει ότι η συνεχόμενη διαδικασία ρύθμισης, η οποία οφειλόταν στη μη συμμόρφωση της κατασκευής με το επιθυμητό αποτέλεσμα οδηγούσε στην αύξηση της διακύμανσης και συνεπώς στην υποβάθμιση της ποιότητας. Ο Shewhart πλαισίωσε το πρόβλημα αυτό με τους όρους των τυχαίων και ειδικών αιτιών διακύμανσης και το 1924 σε ένα εσωτερικό σημείωμα που έγραψε, εισήγαγε το διάγραμμα ελέγχου ως εργαλείο για τη διάκριση μεταξύ αυτών των δύο αιτιών.

Ο Shewhart τόνισε ότι για να μπορέσει μια παραγωγική διαδικασία να βρεθεί σε κατάσταση στατιστικού ελέγχου θα πρέπει να υπάρχουν μόνο τυχαία αίτια μεταβολής. Επίσης, η διατήρηση της κατάστασης ελέγχου της παραγωγικής διαδικασίας είναι απαραίτητη για την πρόβλεψη μελλοντικών αποτελεσμάτων της, τα οποία θα επιφέρουν και καλύτερο οικονομικό αποτέλεσμα στην επιχείρηση.

4.1 Περιγραφή Διαγραμμάτων Ελέγχου

Τα διαγράμματα ελέγχου όπως αναφέραμε και στο προηγούμενο κεφάλαιο είναι ένα από τα κύρια εργαλεία του Στατιστικού Ελέγχου Διεργασιών. Η βασική ιδέα ενός διαγράμματος ελέγχου είναι να διαπιστωθεί αν ισχύει η υπόθεση ότι υπάρχουν μόνο τυχαίες αιτίες μεταβλητότητας έναντι της εναλλακτικής ότι υπάρχουν και ειδικές αιτίες. Αυτό πραγματοποιείται με την ανίχνευση σε πραγματικό χρόνο της εμφάνισης ειδικών αιτιών

μεταβλητότητας σε μια διεργασία και ως συνέπεια τη διαπίστωση της κατάστασης ελέγχου της. Η μορφή που έχει ένα διάγραμμα ελέγχου φαίνεται στο επόμενο σχήμα.



Σχήμα 8: *Διάγραμμα Shewhart*

Τα διαγράμματα ελέγχου είναι κατάλληλα για εφαρμογή σε ανεξάρτητα δεδομένα που πανομοιότυπα ακολουθούν την κανονική κατανομή. Η ευρεία χρήση των διαγραμμάτων ελέγχου καθώς και η δημοτικότητά τους είναι αποτέλεσμα πολλών λόγων. Ένας από αυτούς είναι το γεγονός ότι απεικονίζουν γραφικά την κατάσταση μιας διεργασίας. Έτσι, δίνεται η δυνατότητα ακόμη και σε κάποιον που δεν έχει γνώσεις στατιστικής να κρίνει αν η διεργασία που απεικονίζεται είναι υπό έλεγχο.

Κατά τη διάρκεια των 80 και πλέον χρόνων ζωής των διαγραμμάτων ελέγχου, έχουν σχεδιαστεί και έχουν εφαρμοστεί πολλοί τύποι διαγραμμάτων ελέγχου, που διαφέρουν ως προς τη μορφή τους, το είδος της χαρακτηριστικής παραμέτρου που ελέγχουν (ιδιότητα ή μετρήσιμο) καθώς και της στατιστικής συνάρτησης που χρησιμοποιούν για τον έλεγχο αυτό (μέση τιμή, διασπορά, κ.λπ.). Παρ΄ όλες τις διαφορές τους, υπάρχουν ορισμένα κοινά στοιχεία και βασικές αρχές που χαρακτηρίζουν το σύνολο των διαγραμμάτων ελέγχου. Οι τρεις πιο σημαντικοί τύποι διαγραμμάτων για μεταβλητές, είναι:

α) Τα διαγράμματα ελέγχου Shewhart,

β) Τα διαγράμματα εκθετικά σταθμισμένων κινητών μέσων (EWMA) και

γ) Τα διαγράμματα CUSUM.

Όπως φαίνεται και από το Σχήμα 8, ένα διάγραμμα ελέγχου αποτελείται από τα εξής στοιχεία:

- μια σειρά μετρήσεων/δεδομένων που ακολουθούν μια χρονική σειρά (τουλάχιστον 20-25 σημεία)
- τρεις οριζόντιες γραμμές που αντιστοιχούν:
  - στο κάτω όριο ελέγχου (Lower Control Limit: LCL)
  - στην κεντρική γραμμή (Center Line) και
  - στο άνω όριο ελέγχου (Upper Control Limit: UCL)

Η κεντρική γραμμή CL ενός διαγράμματος ελέγχου αντιστοιχεί στην τιμή της παραμέτρου (π.χ. ποσοστό ελαττωματικών, μέση τιμή) όταν η διεργασία βρίσκεται σε στατιστικό έλεγχο. Εκφράζει, δηλαδή την κατάσταση ομαλής λειτουργίας, όπου η μεταβλητότητα οφείλεται μόνο σε τυχαίες αιτίες.

Τα δύο όρια ελέγχου LCL και UCL αντιστοιχούν στο αναμενόμενο εύρος της μεταβλητότητας βάσει της διεργασίας. Εάν όλα τα σημεία του διαγράμματος βρίσκονται μεταξύ των δύο ορίων ελέγχου και δεν παρουσιάζουν κάποια συστηματική μορφή, η διεργασία είναι εντός ελέγχου. Στην αντίθετη περίπτωση που κάποιο σημείο του διαγράμματος είναι εκτός των ορίων ελέγχου, τότε αυτό αποτελεί ένδειξη ότι η διεργασία μπορεί να βρίσκεται σε κατάσταση εκτός στατιστικού ελέγχου (συναγερμός). Η συμπεριφορά αυτή ερευνάται για να ανακαλυφθούν οι ειδικές αιτίες μεταβλητότητας εφόσον υπάρχουν και αν κριθεί απαραίτητο να αποκατασταθεί η ευστάθεια της διεργασίας με κατάλληλες διορθωτικές ενέργειες. Η επιλογή του εύρους των ορίων ελέγχου αποτελεί το σημαντικότερο σημείο για το σχεδιασμό των διαγραμμάτων ελέγχου.

Σε αυτό το σημείο θα πρέπει να σημειωθεί ότι ακόμη και στην περίπτωση που όλα τα σημεία βρίσκονται εντός των ορίων ελέγχου δεν σημαίνει απαραίτητα ότι η διεργασία είναι εντός στατιστικού ελέγχου. Υπάρχουν περιπτώσεις στις οποίες η διάταξη των σημείων εντός των ορίων να αποκαλύπτει την παρουσία μιας ειδικής αιτίας. Για το λόγο αυτό τα διαδοχικά σημεία ενώνονται μεταξύ τους με ευθύγραμμα τμήματα, ώστε να γίνονται ευκολότερα αντιληπτές οι διατάξεις που μπορεί να κρύβουν κάποια ειδική αιτία. Όταν μια διεργασία είναι πραγματικά εντός στατιστικού ελέγχου τα σημεία του διαγράμματος εμφανίζονται κατά απολύτως τυχαίο τρόπο μέσα στα όρια με το μεγαλύτερο ποσοστό τους να βρίσκεται κοντά στην κεντρική γραμμή και από τις δύο πλευρές της.

4.2 Κατηγορίες Διαγραμμάτων Ελέγχου

Στα διαγράμματα ελέγχου μπορούμε να διακρίνουμε δύο βασικές κατηγορίες ανάλογα με το είδος της μεταβλητής που περιγράφει το ποιοτικό χαρακτηριστικό του προϊόντος:

- Διαγράμματα ελέγχου μεταβλητών (control charts for variables), τα οποία χρησιμοποιούνται όταν τα χαρακτηριστικά που εξετάζονται είναι συνεχή και μετρήσιμα, όπως το βάρος, η θερμοκρασία, ο όγκος κ.λπ. Κυριότεροι εκπρόσωποι αυτής της κατηγορίας είναι τα διαγράμματα μέσης τιμής $\bar{X} - chart$, εύρους $R - chart$ και το διάγραμμα τυπικής απόκλισης $\sigma - chart$.

- Διαγράμματα ελέγχου χαρακτηριστικών ή ιδιοτήτων (control charts for attributes), τα οποία χρησιμοποιούνται όταν τα δεδομένα είναι διακριτά και μη μετρήσιμα (π.χ. αποδεκτό-απορριπτέο). Κυριότεροι εκπρόσωποι είναι τα διαγράμματα $p - chart$ (συχνότητα απορριπτέων ανά δείγμα), $np - chart$ (συχνότητα απορριπτέων ανά σταθερό δείγμα), $c - chart$ (αριθμός απορριπτέων ανά μονάδα επιθεώρησης) σταθερού μεγέθους) και $u - chart$ (μέσος αριθμός απορριπτέων ανά επιθεωρούμενη μονάδα).

4.3 Μαθηματική Περιγραφή Διαγραμμάτων Ελέγχου

Από μαθηματική (στατιστική) άποψη ένα διάγραμμα ελέγχου είναι η γραφική παράσταση μιας παραμέτρου, που αποτελεί χαρακτηριστική ιδιότητα μιας διεργασίας, σε συνάρτηση με το χρόνο ή τον αριθμό δείγματος. Η παράμετρος αυτή, όπως αναφέραμε και παραπάνω, μπορεί να είναι το μήκος, το βάρος, ο όγκος, ή πιο γενικά μια συνεχής και μετρήσιμη μεταβλητή, η οποία συνήθως ακολουθεί μια τυχαία κατανομή. Η κατανομή αυτή μπορεί να βρεθεί μέσω τυχαίων δειγμάτων της μεταβλητής, που λαμβάνονται κατά διαστήματα από την παραγωγική διεργασία που παρακολουθείται. Στη συνέχεια, περιγράφεται η διαδικασία που ακολουθείται για την κατασκευή ενός διαγράμματος Shewhart.

Έστω, λοιπόν, ότι κατά την εκτέλεση μιας διεργασίας παρακολουθείται η τυχαία μεταβλητή X. Κατά τη διάρκεια του χρόνου συλλέγονται n $(n \geq 1)$ δείγματα της μεταβλητής X, τα οποία ας τα συμβολίσουμε ως $X_1, X_2, ..., X_n$. Στη συνέχεια, χρησιμοποιώντας αυτά τα τυχαία δείγματα υπολογίζεται η τιμή μιας κατάλληλης στατιστικής συνάρτησης W της τυχαίας μεταβλητής X, δηλαδή $W_i = g(X_i), \quad i = 1, 2, ..., n$, η οποία απεικονίζεται σε ένα διάγραμμα ως συνάρτηση του χρόνου t ή του μεγέθους των δειγμάτων n.

Η συνάρτηση αυτή εκτιμά την ποσότητα που μας ενδιαφέρει, η οποία μπορεί να είναι για παράδειγμα η μέση τιμή $(\bar{X})$ ή η διακύμανση της X $(var(X))$. Έτσι, η παρακολούθηση της κρίσιμης ποσότητας επιτυγχάνεται ουσιαστικά με την παρακολούθηση των τιμών που λαμβάνει η στατιστική συνάρτηση W στα διάφορα δείγματα X_i.

Για παράδειγμα, ας υποθέσουμε ότι μας ενδιαφέρει η συμπεριφορά της μέσης τιμής του X, όπου X είναι η θερμοκρασία του περιβάλλοντος. Έστω ότι έχουμε ένα δείγμα από n $(n \geq 1)$ μετρήσεις, οι οποίες έχουν ληφθεί σε διαφορετικά χρονικά διαστήματα. Η στατιστική συνάρτηση που μπορούμε να χρησιμοποιήσουμε σε αυτή την περίπτωση, λοιπόν, είναι η $W_i = g(X_i) = (X_1 + X_2 + ... + X_n)/n$, με την οποία υπολογίζεται η μέση τιμή ενός οποιουδήποτε φυσικού μεγέθους.

Στο διάγραμμα που έχει σχεδιαστεί η στατιστική συνάρτηση $W_i = g(X_i)$ πρέπει να καθοριστούν και στη συνέχεια να σχεδιαστούν οι τρεις χαρακτηριστικές γραμμές, που

είναι η κεντρική γραμμή CL και τα δυο όρια ελέγχου, το άνω όριο ελέγχου UCL και το κάτω όριο ελέγχου LCL. Η κεντρική γραμμή, όπως έχουμε αναφέρει και προηγουμένως απεικονίζει τη μέση τιμή της στατιστικής συνάρτησης και καθορίζει το μέσο επίπεδο της διεργασίας χωρίς την παρουσία ειδικών αιτιών μεταβλητότητας (εντός ελέγχου διεργασία).

4.3.1 Όρια Ελέγχου και Κεντρική Γραμμή

Το γενικό μοντέλο δημιουργίας ορίων κατά την κατασκευή ενός διαγράμματος ελέγχου ονομάζεται *μοντέλο ορίων σίγμα* (sigma limits model) και περιγράφεται από τις παρακάτω σχέσεις:

$$\begin{aligned} UCL &= \mu_W + L\sigma_W \\ CL &= \mu_W \\ LCL &= \mu_W - L\sigma_W \end{aligned} \tag{4.3.1}$$

Τα μ_W και σ_W δηλώνουν τη μέση τιμή και την τυπική απόκλιση της στατιστικής συνάρτησης W αντίστοιχα. Ο αριθμός L δηλώνει την απόσταση των ορίων ελέγχου από την κεντρική γραμμή CL σε μονάδες τυπικής απόκλισης. Όπως αναφέραμε και σε προηγούμενο κεφάλαιο όταν ένα σημείο βρεθεί εκτός των ορίων ελέγχου τότε έχουμε ένδειξη ότι η διεργασία βρίσκεται εκτός στατιστικού ελέγχου.

Το μοντέλο αυτό δημιουργήθηκε από τον Shewhart, ο οποίος πρότεινε η τιμή του L να είναι ίση με τρεις τυπικές αποκλίσεις, δηλαδή $L = 3\sigma$. Έτσι, σύμφωνα και με το θεώρημα κεντρικού ορίου, το 99.7% των τιμών της στατιστικής συνάρτησης W θα κατανέμεται μέσα σε μια ζώνη $\pm 3\sigma$. Σχεδιάζοντας, λοιπόν, τα όρια σε αυτές τις τιμές είναι εφικτή η ανίχνευση σημαντικών αλλαγών στη συμπεριφορά μιας διεργασίας επιτυγχάνοντας έτσι και μια εξισορρόπηση μεταξύ των δύο τύπων σφαλμάτων:

Σφάλμα Τύπου I: συμβαίνει όταν τα όρια σχεδιάζονται πολύ στενά, οπότε μια τυχαία αιτία μεταβλητότητας θεωρείται ως ειδικό αίτιο.

Σφάλμα Τύπου ΙΙ: συμβαίνει εάν τα όρια έχουν αρκετά μεγάλο εύρος, οπότε ένα ειδικό αίτιο θεωρείται ως τυχαίο.

Εκτός από την πρόταση που έκανε αρχικά ο Shewhart, προτάθηκαν αργότερα και άλλες τιμές για το L με σκοπό να καλυφθούν και οι διαφορετικές απαιτήσεις της κάθε διεργασίας στην πράξη.

Σε αυτό το σημείο ας συνοψίσουμε όσα έχουμε αναφέρει και αφορούν την κατασκευή ενός διαγράμματος ελέγχου. Για την ανάπτυξη των διαγραμμάτων ελέγχου Shewhart, αρκεί η γνώση της κατανομής της απεικονιζόμενης στατιστικής συνάρτησης W κι έπειτα η κατάλληλη επιλογή για την τιμή του L, που καθορίζει τα όρια ελέγχου. Το διάγραμμα που κατασκευάζεται κατά αυτόν τον τρόπο χρησιμοποιείται για να ελέγξει αν η υπό παρακολούθηση διεργασία παραμένει εντός στατιστικού ελέγχου. Η ύπαρξη τιμών της W που εμφανίζονται έξω από τα όρια ελέγχου, όπως και η μη τυχαία κατανομή των δεδομένων εντός των ορίων ελέγχου, αποτελούν ενδείξεις μιας εκτός ελέγχου διεργασίας, οπότε απαιτείται διερεύνηση των ειδικών αιτών μεταβλητότητας.

Ένα μειονέκτημα των διαγραμμάτων ελέγχου Shewhart είναι ότι δεν έχουν μνήμη και συνεπώς δεν είναι ευαίσθητα στην ανίχνευση μικρών ή και μεσαίων μεταβολών στην τιμή της υπό παρακολούθησης μεταβλητής. Για το λόγο αυτό, έχουν προταθεί διαγράμματα τα οποία χρησιμοποιούν σύνθετους κανόνες για την ανακήρυξη μιας διεργασίας εκτός ελέγχου. Ειδικότερα, ένας τρόπος για να αυξηθεί η ευαισθησία του διαγράμματος είναι η χρήση προειδοποιητικών ορίων ελέγχου, όπως περιγράφεται στην επόμενη παράγραφο. Εκτός, όμως, από την τεχνική αυτή που χρησιμοιποιεί επιπλέον όρια ελέγχου στα διαγράμματα Shewhart, υπάρχουν και τα διαγράμματα CUSUM, τα οποία σε αντίθεση με τα πρώτα έχουν μνήμη και περιγράφονται πιο αναλυτικά στο Κεφάλαιο 5.

4.3.2 Προειδοποιητικά Όρια Ελέγχου

Σε πολλές περιπτώσεις για να γίνει ένα διάγραμμα ελέγχου περισσότερο ευαίσθητο ως προς την ικανότητά του να ανιχνεύει εγκαίρως εκτός ελέγχου διεργασίες, εκτός από

τα όρια ελέγχου, σχεδιάζονται και τα προειδοποιητικά όρια εσωτερικά των ακραίων ορίων ελέγχου.

Τα προειδοποιητικά όρια χρησιμοποιούνται μαζί με κάποιους πρόσθετους κανόνες που έχουν αναπτυχθεί για να περιγράψουν ενδεχόμενα σχετικά με την εμφάνιση ειδικών ακολουθιών σημείων σ΄ ένα διάγραμμα ελέγχου. Στην περίπτωση που συμβεί το ενδεχόμενο που περιγράφει ο κανόνας, τότε η διεργασία θεωρείται εκτός ελέγχου, χωρίς απαραίτητα να έχουμε κάποιο σημείο του διαγράμματος εκτός των ορίων ελέγχου. Η σωστή χρήση των διαγραμμάτων ελέγχου αποφέρει τα ακόλουθα άμεσα και έμμεσα οφέλη που δικαιολογούν την ευρύτατη εφαρμογή τους:

- Μείωση του κόστους εσωτερικών αστοχιών και αύξηση της παραγωγικότητας χάρις στον έγκαιρο εντοπισμό προβλημάτων της διαδικασίας και την αποφυγή παραγωγής ελαττωματικών προϊόντων.
- Βελτίωση της απόδοσης και ποιότητας της παραγωγικής διαδικασίας μέσω της διάγνωσης προβλημάτων που οφείλονται σε κακό σχεδιασμό της διαδικασίας.
- Αποφυγή υπερβολικά συχνών και όχι απαραίτητων επεμβάσεων στην παραγωγική διαδικασία.
- Αξιόπιστη ανάλυση των δυνατοτήτων μιας παραγωγικής διαδικασίας.
- Συστηματική παρακολούθηση των διαδικασιών, καταγραφή μετρήσεων και δημιουργία αρχείου με χρήσιμα στοιχεία και πληροφορίες για την αξιολόγηση των διαδικασιών (αρχεία ποιότητας κατά ISO 9001).
- Έμφαση στην πρόληψη δημιουργίας ελαττωματικών προϊόντων.

4.3.3 Μέτρο απόδοσης ενός διαγράμματος ελέγχου

Σε ένα διάγραμμα ελέγχου έχουμε δύο στόχους. Πρώτον, όταν μια διεργασία είναι εντός ελέγχου θέλουμε το διάγραμμά μας να μη δίνει εσφαλμένους συναγερμούς ή έστω να

δίνει πολύ σπάνια ένα ψευδή συναγερμό. Από πλευρά στατιστικής θέλουμε το διάγραμμά μας να λειτουργεί με την προβλεπόμενη πιθανότητα του ενδεχομένου να σχεδιαστεί ένα σημείο της στατιστικής συνάρτησης εκτός ορίων ελέγχου ενώ η διεργασία να βρίσκεται σε έλεγχο. Δεύτερον, όταν μια διεργασία είναι εκτός ελέγχου θέλουμε το διάγραμμά μας να το σηματοδοτεί εγκαίρως. Στατιστικά, δηλαδή, θέλουμε η πιθανότητα του ενδεχομένου να σχεδιαστεί ένα σημείο εντός των ορίων ελέγχου ενώ η διεργασία να βρίσκεται εκτός ελέγχου να είναι πολύ μικρή.

Για την αξιολόγηση της απόδοσης ενός διαγράμματος ελέγχου, η οποία σχετίζεται και με τους δύο στόχους που αναφέραμε παραπάνω, έχουν προταθεί διάφοροι δείκτες. Ο πιο γνωστός από αυτούς είναι το *μέσο μήκος ροής* (ή μέσο μήκος διαδρομής) του διαγράμματος (Average Run Length, ARL) που ορίζεται με τη σχέση,

$$ARL = \frac{1}{p} \tag{4.3.2}$$

όπου p συμβολίζει την πιθανότητα να βρεθεί ένα σημείο του διαγράμματος ελέγχου εκτός των ορίων ελέγχου. Το ARL σύμφωνα και με τον E. S. Page (1954) είναι ο αναμενόμενος αριθμός (μέση τιμή) των σημείων (δειγμάτων) που πρέπει να σχεδιαστούν σε ένα διάγραμμα ελέγχου για να εμφανιστεί ένα σημείο εκτός των ορίων ελέγχου.

Έχει αποδειχθεί μέσω υπολογισμού του ARL ότι τα διαγράμματα CUSUM είναι πιο αποδοτικά σε σύγκριση με τα διαγράμματα Shewhart όταν απαιτείται η ανίχνευση μικρών μεταβολών στη μέση τιμή της χαρακτηριστικής μεταβλητής. Συγκεκριμένα όταν οι μεταβολές αυτές είναι μικρότερες από 2 τυπικές αποκλίσεις ($< 2\sigma$).

5 Διαγράμματα CUSUM

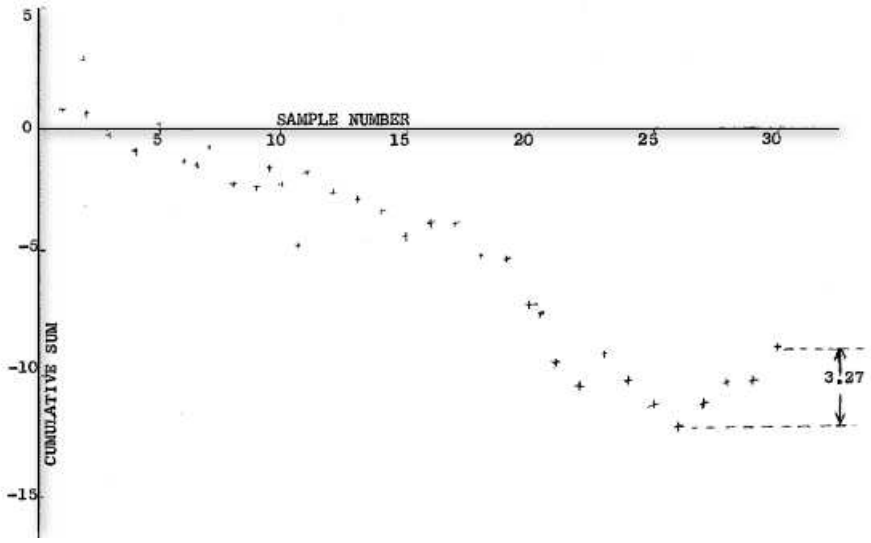
Τα διαγράμματα CUSUM (Cumulative Sum Charts) επινοήθηκαν και προτάθηκαν από τον E. S. Page (1954) και μπορούν να χρησιμοποιηθούν τόσο για τον έλεγχο συνεχών μεταβλητών όσο και για τον έλεγχο διακριτών μεταβλητών. Τα διαγράμματα αυτά είναι ένα από τα ισχυρότερα διαθέσιμα εργαλεία για την ανίχνευση των μικρών αλλαγών, επειδή λαμβάνονται υπόψη όλα τα ιστορικά δεδομένα. Ειδικότερα, τα διαγράμματα ελέγχου τύπου CUSUM χαρακτηρίζονται ως διαγράμματα με ομοιόμορφη μνήμη αφού δίνουν την ίδια βαρύτητα σε όλες τις προηγούμενες παρατηρήσεις.

Η μέθοδος που πρότεινε ο Page βασίζεται στον υπολογισμό ενός συσσωρευτικού αθροίσματος το οποίο προκύπτει από την ακόλουθη διαδικασία: Αρχικά, επιλέγονται m δείγματα καθένα από τα οποία περιέχει n μετρήσεις της μεταβλητής x, όπου η x μπορεί να αντιπροσωπεύει ένα χαρακτηριστικό μέγεθος κατά την εκτέλεση μιας διεργασίας. Αν συμβολίσουμε με $\bar{x}_i$ τη μέση τιμή του i δείγματος (δειγματικός μέσος), τότε το συσσωρευτικό άθροισμα S_m υπολογίζεται μέσω μίας εκ των δύο παρακάτω σχέσεων:

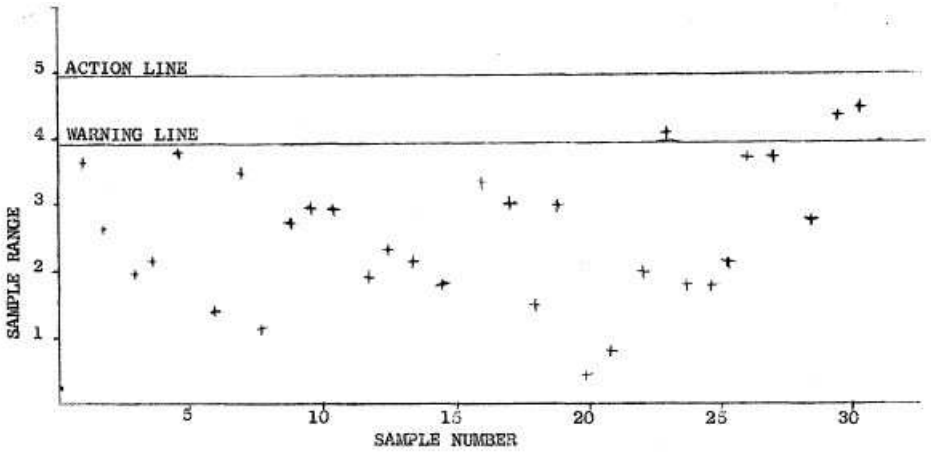
$$S_m = \sum_{i=1}^{m} (\bar{x}_i - \hat{\mu}_0) \tag{5.0.3}$$

$$S'_m = \frac{1}{\sigma_{\bar{x}}} \sum_{i=1}^{m} (\bar{x}_i - \hat{\mu}_0) \tag{5.0.4}$$

όπου το $\hat{\mu}_0$ συμβολίζει την εκτιμώμενη μέση τιμή της μεταβλητής x όταν η διεργασία βρίσκεται εντός ελέγχου και το $\sigma_{\bar{x}}$ συμβολίζει την γνωστή (ή εκτιμώμενη πάλι) τυπική απόκλιση των δειγματικών μέσων $\bar{x}_i$. Το διάγραμμα CUSUM είναι η απεικόνιση του αθροίσματος S_m ή S'_m (βλ.Σχήμα 9 και Σχήμα 10) σε συνάρτηση με τον αριθμό των δειγμάτων m και στην περίπτωση που $S_m > h$ ή $S'_m > h/\sigma$ ο αλγόριθμος CUSUM σηματοδοτεί ένα συναγερμό. Το h είναι το όριο ελέγχου (decision interval) σε ένα διάγραμμα CUSUM και η τιμή του καθορίζεται ανάλογα με το εκάστοτε πρόβλημα που μελετάται.



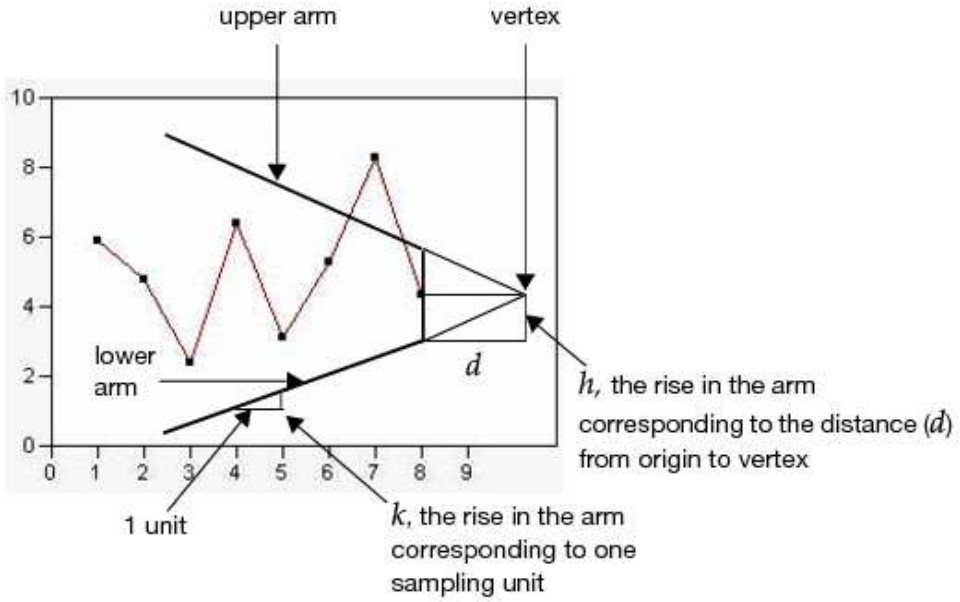
Σχήμα 9: *Διάγραμμα CUSUM, Page (1963)*



Σχήμα 10: *Διάγραμμα CUSUM με προειδοποιητικό όριο, Page (1963)*

Λίγα χρόνια μετά από την πρόταση του Page, το 1959 ο Barnard πρότεινε μία παραλλαγή του CUSUM, η οποία είναι γνωστή ως μέθοδος *V-mask CUSUM*. Η μέθοδος αυτή υπολογίζει όπως περιγράφηκε παραπάνω τα συσσωρευτικά αθροίσματα απλά διαφέρει στον τρόπο που θα εντοπίσει αν η διεργασία είναι εντός ή εκτός ελέγχου. Στο τελευταίο συσσωρευτικό άθροισμα που έχει υπολογιστεί εφαρμόζεται μια μάσκα σχήματος V και στην περίπτωση που όλα τα προηγούμενα σημεία βρίσκονται εντός της περιοχής της μάσκας τότε η διεργασία βρίσκεται εντός ελέγχου, ενώ στην αντίθετη περίπτωση που κάποιο από

τα προηγούμενα σημεία βρεθεί εκτός της περιοχής της μάσκας τότε η διεργασία κρίνεται εκτός ελέγχου. Στο παρακάτω διάγραμμα του σχήματος 11, φαίνεται μία εκτός ελέγχου διεργασία η οποία έχει εντοπιστεί με την εφαρμογή της μεθόδου V-mask CUSUM.



Σχήμα 11: *Διάγραμμα V − mask CUSUM*

Η μέθοδος V-mask CUSUM χρησιμοποιήθηκε ιδιαίτερα τα χρόνια πριν την εμφάνιση των ηλεκτρονικών υπολογιστών, ενώ μετά αναπτύχθηκε μια απλούστερη παραλλαγή της μεθόδου CUSUM, η λεγόμενη μέθοδος *tabular (algorithmic) CUSUM*. Με την πάροδο των χρόνων, φαίνεται να έχει επικρατήσει η μέθοδος tabular CUSUM κυρίως λόγω της απλότητάς της σε σχέση με την μέθοδο V-mask. Στη συνέχεια δίνεται η περιγραφή της tabular μεθόδου κατασκευής ενός διαγράμματος CUSUM.

5.1 Περιγραφή Μεθόδου Tabular CUSUM

Ας υποθέσουμε αρχικά ότι κατά την εκτέλεση μιας διεργασίας είναι υπό παρακολούθηση η συνεχής μεταβλητή x, η οποία αποτελεί κάποιο χαρακτηριστικό της διεργασίας που εξετάζεται. Επιπλέον, η μεταβλητή αυτή θα προσδιορίσει την κατάσταση στην οποία θα βρίσκεται η διεργασία ανά πάσα στιγμή (στατιστικός έλεγχος).

Έστω, ότι τις χρονικές στιγμές $t_i = 1, \cdots, m$ έχουν συλλεχθεί οι αντίστοιχες μετρήσεις x_i της μεταβλητής x. Θεωρούμε, επίσης, ότι η εντός ελέγχου μέση τιμή μ_0 και η τυπική απόκλιση σ της x έχουν υπολογιστεί μετά από ένα χρονικό διάστημα αναφοράς και είναι γνωστές. Για την κάθε μέτρηση υπολογίζεται η διαφορά:

$$z_i = x_i - \mu_0$$

η οποία ουσιαστικά είναι η απόκλιση της κάθε μέτρησης x_i από την μέση τιμή μ_0. Πολλές φορές χρησιμοποιείται και ο μετασχηματισμός:

$$z_i' = \frac{x_i - \mu_0}{\sigma} \tag{5.1.1}$$

ο οποίος μετατρέπει την απόκλιση z_i σε μονάδες τυπικής απόκλισης σ (κανονικοποίηση). Ο λόγος για την εφαρμογή του μετασχηματισμού 5.1.1 είναι επειδή έχει αποδειχθεί ότι:

> Αν η τυχαία μεταβλητή X ακολουθεί μια κανονική κατανομή $N(\mu, \sigma^2)$, τότε η τυχαία μεταβλητή $Z = (X - \mu)/\sigma$, ακολουθεί την τυποποιημένη κανονική κατανομή $N(0, 1)$ (βλ.παρ.3.4.3).

Για το επιλεγμένο χρονικό διάστημα ο αλγόριθμος του CUSUM υπολογίζει αναδρομικά δύο συσσωρευτικά αθροίσματα. Το ένα από αυτά αφορά τις θετικές αποκλίσεις ("one-sided upper CUSUM"):

$$\begin{aligned} S_0^+ &= 0 \\ S_i^+ &= \max[0, S_{i-1}^+ + (x_i - \mu_0) - K] \end{aligned} \tag{5.1.2}$$

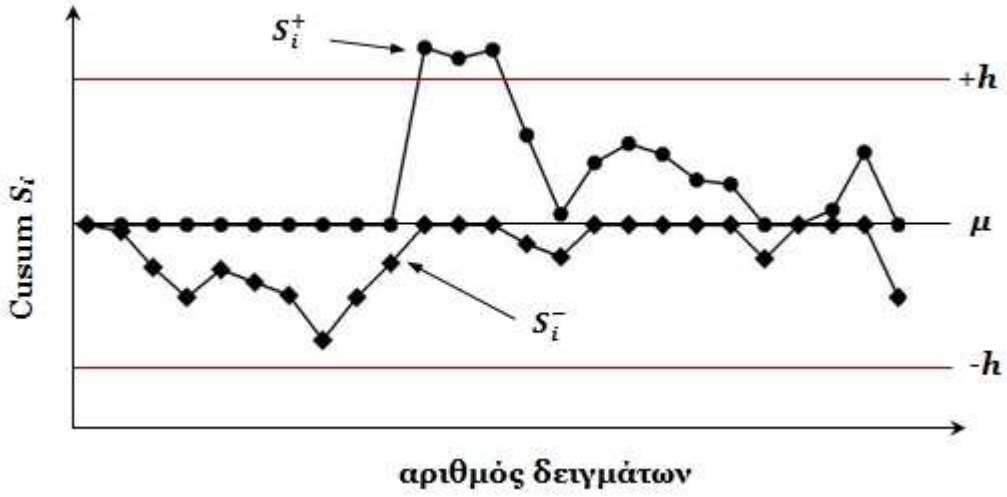
ενώ το δεύτερο αφορά τις αρνητικές αποκλίσεις ("one-sided lower CUSUM"):

$$S_0^- = 0$$
$$S_i^- = \min[0, S_{i-1}^- + (x_i - \mu_0) + K] \tag{5.1.3}$$

όπου $K \geq 0$ είναι μια σταθερά που ονομάζεται *τιμή αναφοράς* (reference value) και τα S_0^+, S_0^- είναι οι *τιμές εκκίνησης* για τα αντίστοιχα συσσωρευτικά αθροίσματα. Το διάγραμμα CUSUM είναι ουσιαστικά η απεικόνιση των δύο παραπάνω αθροισμάτων σε συνάρτηση με το κάθε δείγμα i. Ο αλγόριθμος CUSUM θα σηματοδοτήσει συναγερμό σε μία από τις δύο περιπτώσεις:

$$S_i^+ > h \tag{5.1.4}$$
$$S_i^- < -h \tag{5.1.5}$$

όπου h και $-h$ είναι τα δύο όρια ελέγχου (h: decision interval), τα οποία σχεδιάζονται στο διάγραμμα CUSUM και είναι ευθείες παράλληλες προς την ευθεία που απεικονίζει τη μέση τιμή μ_0 (βλ.Σχήμα 12).



Σχήμα 12: *Διάγραμμα Tabular CUSUM*

Οι τιμές των δύο σταθερών K και h καθορίζονται ανάλογα με το πρόβλημα στο οποίο εφαρμόζεται ο αλγόριθμος CUSUM. Η τιμή αναφοράς K σχετίζεται με το μέγεθος της μικρότερης απόκλισης από τη μέση τιμή που επιθυμείται να ανιχνεύσει ο αλγόριθμος γρήγορα. Η πιο συνηθισμένη σχέση που χρησιμοποιείται για τον υπολογισμό της τιμής αναφοράς K είναι η εξής:

$$K = \frac{|\mu_1 - \mu_0|}{2} = \frac{\delta\sigma}{2}, \quad \delta = \frac{|\mu_1 - \mu_0|}{\sigma} \tag{5.1.6}$$

όπου η τιμή μ_0 δηλώνει μια εντός ελέγχου μέση τιμή, όπως ήδη έχουμε αναφέρει, ενώ η μ_1 δηλώνει μια εκτός ελέγχου μέση τιμή.

Αν γράψουμε τις σχέσεις 5.1.2 και 5.1.3 με την παρακάτω μορφή:

$$S_i^+ = \max[0, S_{i-1}^+ + x_i - (\mu_0 + K)]$$
$$S_i^- = \min[0, S_{i-1}^- + x_i - (\mu_0 - K)]$$

θα δούμε ότι η ποσότητα S_i^+ συσσωρεύει τις αποκλίσεις των παρατηρήσεων x_i από την ποσότητα $(\mu_0 + K)$ από τη στιγμή που θα εμφανιστεί μια θετική απόκλιση και μπορεί να θεωρηθεί κατάλληλη για τον έλεγχο της υπόθεσης:

$$H_0 : \mu = \mu_0$$

έναντι της υπόθεσης

$$H_1^+ : \mu = \mu_1 = \mu_0 + \delta\sigma$$

αφού μεγάλες θετικές τιμές της S_i^+ οδηγούν σε αποδοχή της υπόθεσης H_1^+. Αντίστοιχα, η ποσότητα S_i^- συσσωρεύει τις αποκλίσεις των παρατηρήσεων x_i από την ποσότητα $(\mu_0 - K)$ από τη στιγμή που θα εμφανιστεί αρνητική απόκλιση και μπορεί να θεωρηθεί

κατάλληλη για τον έλεγχο της υπόθεσης:

$$H_0 : \mu = \mu_0$$

έναντι της υπόθεσης

$$H_1^- : \mu = \mu_1 = \mu_0 - \delta\sigma$$

αφού μεγάλες αρνητικές τιμές της S_i^- οδηγούν σε αποδοχή της υπόθεσης H_1^-. Για το ποιά από τις δύο εναλλακτικές υποθέσεις H_1^+ και H_1^- θα αποδεχθούμε ή όχι σε κάθε βήμα της διαδικασίας θα εξαρτηθεί από το ποιά από τις δύο σχέσεις τελικώς θα ισχύσει, η 5.1.4 ή η 5.1.5. Οποιαδήποτε από τις δύο και αν ισχύσει είναι προφανές ότι θα έχουμε ένδειξη ότι η διεργασία είναι εκτός ελέγχου (μετατόπιση της μέσης τιμής της παραγωγικής διεργασίας σε υψηλότερο ή χαμηλότερο επίπεδο).

Τα διαγράμματα CUSUM έχουν επικρατήσει σε σχέση με τα διαγράμματα Shewhart για το λόγο της απλότητας τους, της μνήμης τους, και της ευαισθησίας στον εντοπισμό μικρών σφαλμάτων. Στην περίπτωση που εμπλέκονται πολλές μεταβλητές με συσχέτιση μεταξύ τους μπορεί να γίνει εφαρμογή των πολυμεταβλητών διαγραμμάτων CUSUM, τα οποία περιγράφονται σε επόμενη παράγραφο. Στη συνέχεια, γίνεται μια μικρή αναφορά στη στατιστική του Hotelling, που αποτελεί την αφετηρία των πολυμεταβλητών διαγραμμάτων ελέγχου.

5.2 Στατιστική του Hotelling

Η πρώτη έρευνα στον Πολυμεταβλητό Στατιστικό Έλεγχο Ποιότητας έγινε από τον Harold Hotelling το 1947. Ο Hotelling εισήγαγε ένα στατιστικό στοιχείο, το οποίο ονόμασε Hotelling's T^2 (Hotelling's T-squared statistic) και το χρησιμοποίησε για να συνδυάσει τις πληροφορίες που έπαιρνε από τη διακύμανση και τη μέση τιμή πολλών μεταβλητών (πολυμεταβλητός έλεγχος υποθέσεων).

Η στατιστική T^2 του Hotelling είναι ένα μέτρο με το οποίο μελετάται η συνδιακύ-

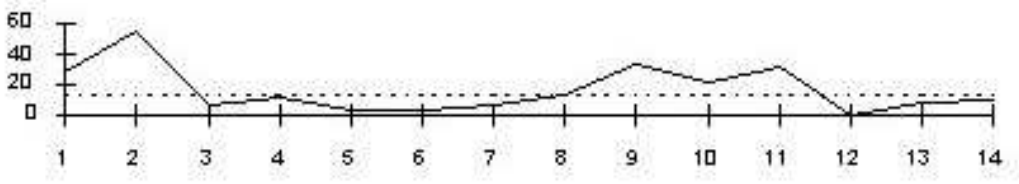
μανση μιας πολυμεταβλητής κανονικής κατανομής και αποτελείται από τις τρεις ακόλουθες ποσότητες:

1. Το διάνυσμα των αποκλίσεων μεταξύ των μετρήσεων και της μέσης τιμής $(\mathbf{x} - \boldsymbol{\mu})$
2. τον αντίστροφο πίνακα συνδιακύμανσης $\boldsymbol{\Sigma}^{-1}$
3. Το ανάστροφο διάνυσμα των αποκλίσεων μεταξύ των μετρήσεων και της μέσης τιμής $(\mathbf{x} - \boldsymbol{\mu})'$

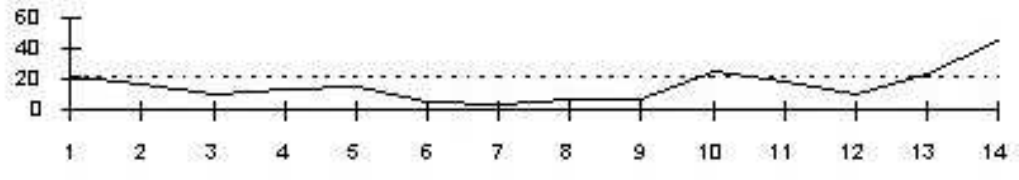
Δηλαδή,

$$T^2 = +(\mathbf{x} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) \tag{5.2.1}$$

Το πολυμεταβλητό διάγραμμα που κατασκευάζεται από τη στατιστική T^2 θα δώσει ένδειξη ότι η διεργασία είναι εκτός ελέγχου όταν $T^2 > SCL$, όπου SCL είναι το όριο ελέγχου του Shewhart (Shewhart Control Limit). Παρακάτω, δίνονται δύο διαγράμματα (Σχήμα 13 και Σχήμα 14) ως παραδείγματα για τη μορφή που έχει η πολυμεταβλητή στατιστική του Hotelling.



Σχήμα 13: *Διάγραμμα Στατιστικής T^2 – Hotelling για τη μέση τιμή*



Σχήμα 14: *Διάγραμμα Στατιστικής T^2 – Hotelling για τη διασπορά*

5.3 Περιγραφή Πολυμεταβλητού Tabular CUSUM

Ο Ronald B. Crosier (1986, 1988) πρότεινε μία μέθοδο πολυμεταβλητού Tabular CUSUM, την οποία ονόμασε *COT* (CUSUM of the scalars T_n). Σύμφωνα με την μέθοδο COT αν το T_n συμβολίζει ένα βαθμωτό μέγεθος με $n = 1, 2, 3, ...$, τότε, ο αλγόριθμος CUSUM για το μέγεθος αυτό θα εκφράζεται σύμφωνα με την παρακάτω σχέση:

$$S_n = \max(0, S_{n-1} + T_n - k) \tag{5.3.1}$$

όπου $S_0 \geq 0$, $k > 0$ και θα σηματοδοτεί μία εκτός ελέγχου διεργασία στην περίπτωση που ισχύσει $S_n > h$. Η COT μέθοδος είναι ουσιαστικά η πιο γενική μορφή του Tabular CUSUM. Έτσι, αν θέλουμε να χρησιμοποιήσουμε την COT μέθοδο για τον εντοπισμό μικρών αλλαγών στη μέση τιμή μιας τυχαίας μεταβλητής x, τότε η σχέση 5.3.1 διαμορφώνεται ως εξής:

$$S_n = \max(0, S_{n-1} + (x_n - \mu) - k\sigma) \tag{5.3.2}$$

όπου μ είναι η μέση τιμή της x και σ η τυπική της απόκλιση. Αν αντικαταστήσουμε όλα τα βαθμωτα μεγέθη στην προηγούμενη σχέση με διανύσματα, τότε η μονομεταβλητή μέθοδος COT μετατρέπεται στην πολυμεταβλητή μέθοδο CUSUM.

$$\mathbf{S}_n = \max(0, \mathbf{S}_{n-1} + (\mathbf{x}_n - \boldsymbol{\mu}) - \mathbf{k}) \tag{5.3.3}$$

Με την αντικατάσταση, όμως, αυτή παρουσιάζονται δύο προβλήματα που πρέπει να λυθούν για να μπορεί να χρησιμοποιηθεί αυτή η πολυμεταβλητή μέθοδος. Το ένα αφορά την εύρεση του $\mathbf{k}$ και το δεύτερο αφορά τον τρόπο με τον οποίο θα ερμηνευτεί το μέγιστο και μηδενικό διάνυσμα. Στην μονομεταβλητή περίπτωση η ποσότητα $S_n + (x_n - \mu)$ συρρικνώνεται προς το μηδέν κατά k τυπικές αποκλίσεις. Επομένως, για να διατηρηθεί αυτή η συμπεριφορά και στην πολυμεταβλητή περίπτωση θα πρέπει το $\mathbf{k}$ να ικανοποιεί την

σχέση,

$$k^2 = \mathbf{k}'\boldsymbol{\Sigma}^{-1}\mathbf{k}$$

όπου k είναι ουσιαστικά το μέτρο του $\mathbf{k}$ που υπολογίζεται με τη βοήθεια του πίνακα συνδιακύμανσης $\boldsymbol{\Sigma}^{-1}$. Αν με την αφαίρεση του διανύσματος $\mathbf{k}$ στη σχέση 5.3.3 πρέπει να συρρικνώνεται το διάνυσμα $[\mathbf{S}_{n-1} + (\mathbf{x}_n - \boldsymbol{\mu})]$ προς το μηδενικό διάνυσμα $\mathbf{0}$, τότε συμπεραίνουμε ότι τα δύο αυτά διανύσματα θα πρέπει να είναι ομόρροπα. Επομένως, μπορούμε να εκφράσουμε το διάνυσμα $\mathbf{k}$ ως:

$$\mathbf{k} = (k/C_n)(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})$$

όπου C_n είναι το μέτρο του διανύσματος $(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})$. Η μορφή που θα έχει, λοιπόν, ο πολυμεταβλητός αλγόριθμος CUSUM περιγράφεται στον ορισμό 5.1.

Ορισμός 5.1. Αν θεωρήσουμε ότι το C_n συμβολίζει το μέτρο του διανύσματος $(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})$, που υπολογίζεται με τη βοήθεια του πίνακα συνδιακύμανσης, δηλαδή,

$$C_n = [(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})]^{1/2} \tag{5.3.4}$$

τότε για το διάνυσμα $\mathbf{s}_n$ θα ισχύει,

$$\begin{aligned} &\mathbf{s}_n = \mathbf{0}, && \text{αν } C_n \leq k \\ &\mathbf{s}_n = (1 - k/C_n)(\mathbf{s}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu}), && \text{αν } C_n > k \end{aligned} \tag{5.3.5}$$

με $\mathbf{s}_0 = \mathbf{0}$. Έστω, επίσης, ότι

$$Y_n = [\mathbf{s}_n'\boldsymbol{\Sigma}^{-1}\mathbf{s}_n]^{1/2} \tag{5.3.6}$$

τότε ο πολυμεταβλητός αλγόριθμος CUSUM θα δώσει ένδειξη εκτός ελέγχου (συναγερμό) όταν $Y_n > h$.

□

Παράδειγμα 5.1. Έστω ότι έχουμε ένα πρόβλημα δύο μεταβλητών x_1 και x_2, οι οποίες ακολουθούν την διμεταβλητή κανονική κατανομή $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, όπου $\boldsymbol{\mu}$ είναι το διάνυσμα των μέσων και $\boldsymbol{\Sigma}$ είναι ο πίνακας συνδιακύμανσης, τα οποία είναι και τα δύο γνωστά και ίσα με:

$$\boldsymbol{\mu} = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Επίσης, είναι γνωστές και οι δύο σταθερές k και h, και ίσες με 0.5 και 5.5 αντίστοιχα. Θεωρούμε το διάνυσμα $\mathbf{x}$, του οποίου οι στήλες θα περιέχουν τις μετρήσεις της κάθε μεταβλητής, δηλαδή,

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

Έστω, ότι λήφθηκαν οι ακόλουθες 5 μετρήσεις για τις δύο αυτές μεταβλητές:

$$\mathbf{x} = \begin{bmatrix} \text{-1.19} & 0.59 \\ 0.12 & 0.90 \\ \text{-1.69} & 0.40 \\ 0.30 & 0.46 \\ 0.89 & \text{-0.76} \end{bmatrix}$$

Ας δούμε αναλυτικά τη διαδικασία που ακολουθεί ο πολυμεταβλητός αλγόριθμος CUSUM. Για $n = 1$, η σχέση 5.3.4 διαμορφώνεται ως εξής:

$$C_1 = [(\mathbf{s}_0 + \mathbf{x}_1 - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\mathbf{s}_0 + \mathbf{x}_1 - \boldsymbol{\mu})]^{1/2}$$

και επειδή,

$$\mathbf{s}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} -1.19 & 0.59 \end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad \boldsymbol{\Sigma}^{-1} = \begin{bmatrix} 1.33 & \text{-0.67} \\ \text{-0.67} & 1.33 \end{bmatrix}$$

η προηγούμενη σχέση διαμορφώνεται ως εξής:

$$C_1 = \left\{ \begin{bmatrix} -1.19 & 0.59 \end{bmatrix} \begin{bmatrix} 1.33 & -0.67 \\ -0.67 & 1.33 \end{bmatrix} \begin{bmatrix} -1.19 \\ 0.59 \end{bmatrix} \right\}^{1/2} =$$

$$= \left\{ \begin{bmatrix} -1.19 & 0.59 \end{bmatrix} \begin{bmatrix} -1.978 \\ 1.582 \end{bmatrix} \right\}^{1/2} \Rightarrow C_1 = (3.28)^{1/2} = 1.813 \quad (> k = 0.5)$$

Παρόμοια, η σχέση 5.3.5 για $n = 1$ και εφόσον ισχύει ότι $C_1 > 0.5 (= k)$, διαμορφώνεται ως εξής:

$$s_1 = [(\mathbf{s}_0 + \mathbf{x}_1 - \boldsymbol{\mu})(1 - k/C_1) = \begin{bmatrix} -1.19 & 0.59 \end{bmatrix} \left(1 - \frac{0.5}{1.813}\right)$$

$$s_1 = \begin{bmatrix} -0.86 & 0.43 \end{bmatrix}$$

Έτσι, για το Y_1 από τη σχέση 5.3.6 θα ισχύει:

$$Y_1 = [\mathbf{s}_1' \boldsymbol{\Sigma}^{-1} \mathbf{s}_1]^{1/2} = \left\{ \begin{bmatrix} -0.86 & 0.43 \end{bmatrix} \begin{bmatrix} 1.33 & -0.67 \\ -0.67 & 1.33 \end{bmatrix} \begin{bmatrix} -0.86 \\ 0.43 \end{bmatrix} \right\}^{1/2}$$

$$Y_1 = \left\{ \begin{bmatrix} -1.432 & 1.1481 \end{bmatrix} \begin{bmatrix} -0.86 \\ 0.43 \end{bmatrix} \right\}^{1/2} \Rightarrow Y_1 = 1.3135 \quad (< h = 5.5)$$

Ακολουθώντας την ίδια διαδικασία για $n = 2$ παίρνουμε τα εξής αποτελέσματα:

$$C_2 = [(\mathbf{s}_1 + \mathbf{x}_2 - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\mathbf{s}_1 + \mathbf{x}_2 - \boldsymbol{\mu})]^{1/2}$$

$$C_2 = \left\{ \begin{bmatrix} -0.86+0.12 & 0.43+0.90 \end{bmatrix} \begin{bmatrix} 1.33 & -0.67 \\ -0.67 & 1.33 \end{bmatrix} \begin{bmatrix} -0.86+0.12 \\ 0.43+0.90 \end{bmatrix} \right\}^{1/2} =$$

$$= \left\{ \begin{bmatrix} -0.74 & 1.33 \end{bmatrix} \begin{bmatrix} 1.33 & -0.67 \\ -0.67 & 1.33 \end{bmatrix} \begin{bmatrix} -0.74 \\ 1.33 \end{bmatrix} \right\}^{1/2} = \left\{ \begin{bmatrix} -0.74 & 1.33 \end{bmatrix} \begin{bmatrix} -1.8753 \\ 2.2647 \end{bmatrix} \right\}^{1/2}$$

$$\Rightarrow C_2 = (4.39977)^{1/2} = 2.098 \quad (> k = 0.5)$$

Για το s_2, εφόσον ισχύει $C_2 > k(= 0.5)$ θα έχουμε:

$$s_2 = [(\mathbf{s}_1 + \mathbf{x}_2 - \boldsymbol{\mu})(1 - k/C_2) = \begin{bmatrix} -0.74 & 1.33 \end{bmatrix} \left(1 - \frac{0.5}{2.098}\right) = \begin{bmatrix} -0.564 & 1.013 \end{bmatrix}$$

Και τέλος για το Y_2, θα ισχύουν οι επόμενες ισότητες:

$$Y_2 = [\mathbf{s}_2' \boldsymbol{\Sigma}^{-1} \mathbf{s}_2]^{1/2} = \left\{ \begin{bmatrix} -0.564 & 1.013 \end{bmatrix} \begin{bmatrix} 1.33 & -0.67 \\ -0.67 & 1.33 \end{bmatrix} \begin{bmatrix} -0.564 \\ 1.013 \end{bmatrix} \right\}^{1/2}$$

$$Y_2 = \left\{ \begin{bmatrix} -0.564 & 1.013 \end{bmatrix} \begin{bmatrix} -1.43 \\ 1.73 \end{bmatrix} \right\}^{1/2} \Rightarrow Y_2 = 1.6 \quad (< h = 5.5)$$

Παρακάτω δίνεται ένας συγκεντρωτικός πίνακας με όλα τα υπόλοιπα αποτελέσματα.

	Μετρήσεις		Διάνυσμα $\mathbf{s}_n$		
n	x_1	x_2	s_1	s_2	Y_n
1	−1.19	0.59	−0.86	0.43	1.31
2	0.12	0.90	−0.56	1.01	1.60
3	−1.69	0.40	−1.95	1.22	3.20
4	0.30	0.46	−1.40	1.43	2.83
5	0.89	−0.75	−0.30	0.39	0.69

Πίνακας 2: *Αριθμητικό Παράδειγμα Διμεταβλητού CUSUM*

♦

6 Εφαρμογές Αλγορίθμου CUSUM

Στο κεφάλαιο αυτό περιγράφονται οι εφαρμογές του μονομεταβλητού αλγορίθμου CUSUM καθώς και του διμεταβλητού αλγορίθμου CUSUM, οι οποίοι αναπτύχθηκαν εξ΄ ολοκλήρου σε Java και εφαρμόστηκαν πάνω στην πλατφόρμα SunSPOT. Οι εφαρμογές αυτές αναπτύχθηκαν με κίνητρο την ανίχνευση πυρκαγιάς σε ανοιχτό περιβάλλον. Αρχικά, όμως, δίνεται μια σύντομη περιγραφή της πλατφόρμας SunSPOT καθώς και του προσομοιωτικού πρρογράμματος Solarium που χρησιμοποιήθηκε για την εξαγωγή των αποτελεσμάτων.

6.1 SunSPOT

Το πρότζεκτ SunSPOT (Sun Small Programmable Object Technology) αποτελεί ένα στιγμιότυπο μιας έρευνας, η οποία ξεκίνησε από την Sun Labs και βρίσκεται ακόμα σε εξέλιξη. Η Sun Labs από την ίδρυσή της το 1990 λειτουργεί σαν το εργαστήριο για εφαρμοσμένη έρευνα και προηγμένη ανάπτυξη της Sun Microsystems, Inc.. Κάποιοι από τους τομείς πάνω στους οποίους η Sun Labs εργάστηκε ήταν τα ασύγχρονα κυκλώματα, οι οπτικές επικοινωνίες, οι νέες τεχνολογίες του διαδικτύου, οι τεχνολογίες Java και τα δίκτυα υπολογιστών.

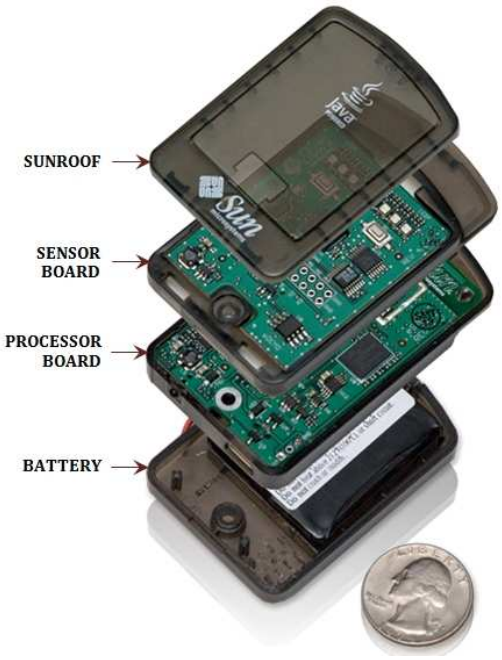
Το 2010 όταν εξαγοράστηκε η Sun από την Oracle, η Sun Labs μετονομάστηκε σε Oracle Labs συνεχίζοντας την αρχική της λειτουργία ως τμήμα πλέον της Oracle Corporation. Η Oracle Labs με τις αναρίθμητες και ποικίλες τεχνολογικές εφευρέσεις έχει εξελιχθεί κατά πολύ σε σχέση με τα πρώτα της βήματα και μέσα σε μικρό χρονικό διάστημα έχει καταφέρει να αναδείξει την Oracle ως μία τεχνολογική υπερδύναμη.

Οι ερευνητές του Sun Labs το 2003 είχαν ξεκινήσει να δουλεύουν πάνω στα ασύρματα δίκτυα αισθητήρων και κατά τη διάρκεια της έρευνάς τους αναπτύχθηκε η ανάγκη για πιο ισχυρές συσκευές αισθητήρων,

οι οποίες θα ήταν πιο εύκολο να προγραμματιστούν. Έτσι, έναν χρόνο μετά γεννήθηκε το πρότζεκτ Sun SPOT με αρχικό στόχο τη δημιουργία κατάλληλου υλικού αισθητήρα, στον οποίο θα μπορούσε να προσαρμοστεί μία ευέλικτη εικονική μηχανή JVM (Java Virtual Machine) πάνω στην πλατφόρμα του αισθητήρα.

Ο σκοπός του πρότζεκτ SunSPOT από την αρχή της δημιουργίας του ήταν να ενθαρρύνει την ανάπτυξη νέων εφαρμογών και συσκευών από προγραμματιστές που δεν είχαν ασχοληθεί ποτέ με ενσωματωμένα συστήματα. Ο σχεδιασμός των συσκευών SunSPOT είναι τέτοιος που επιτρέπει την σύνταξη και την ανάπτυξη προγραμμάτων με τη δυνατότητα να αλληλεπιδρούν μεταξύ τους, με το περιβάλλον καθώς και με τους χρήστες με εντελώς νέους και διαφορετικούς τρόπους.

6.1.1 Περιγραφή της πλατφόρμας Sun SPOT



Σχήμα 15: *Ανατομία ενός SunSPOT*

Το SunSPOT είναι μία Java προγραμματιζόμενη ενσωματωμένη συσκευή που έχει σχεδιαστεί με τέτοιο τρόπο, ώστε να προσφέρει ευελιξία στον χρήστη. Κάθε συσκευή SunSPOT έχει εγκατεστημένη την εικονική μηχανή Squawk, η οποία είναι μία μικρού μεγέθους έκδοση της πρότυπης εικονικής μηχανής της Java. Τα SunSPOTs, λοιπόν, δεν έχουν κάποιο λειτουργικό σύστημα, αλλά τρέχουν την Squawk VM απευθείας πάνω στον επεξεργαστή, και η VM παρέχει τις βασικότερες λειτουργίες ενός λειτουργικού συστήματος.

Η συγκεκριμένη έκδοση Squawk χρησι-

μοποιείται κυρίως σε φορητές και ενσωματω-μένες συσκευές, είναι γραμμένη σχεδόν εξ΄ ολοκλήρου σε Java, εκτός από κάποια μικρά κομμάτια τα οποία έχουν να κάνουν με τον πρωτογενή κώδικα και την είσοδο και έξοδο δεδομένων. Το γεγονός αυτό την καθιστά μία έκδοση εύκολα μεταφέρσιμη και διαφανώς ενσωματώσιμη με τους πόρους της εφαρμογής, όπως είναι τα αντικείμενα, τα νήματα και οι διεπαφές του λειτουργικού συστήματος.

Κάποια από τα βασικά χαρακτηριστικά της συγκεκριμένης συσκευής είναι ότι οι εφαρμογές εκτελούνται απευθείας από την αστραπιαία μνήμη (flash memory), οι οδηγοί των συσκευών (drivers) είναι και αυτοί γραμμένοι σε Java και η διαχείριση της μπαταρίας είναι αυτόματη προσφέροντας μεγάλους χρόνους λειτουργίας και αναμονής. Λόγω της εφαρμογής της Java, ο προγραμματισμός ενός SunSPOT γίνεται με εκπληκτική ευκολία.

Το υλικό της βασικής μονάδας μίας συσκευής SunSPOT περιλαμβάνει αναλυτικά τα εξής:

- ◇ έναν επεξεργαστή ARM920T (180 MHz)
- ◇ μία μνήμη 512K RAM, 4MB Flash
- ◇ μία ασύρματη διεπαφή IEEE 802.15.4 (2.4 GHz)
- ◇ μία θύρα USB
- ◇ ένα επιταχυνσιόμετρο τριών αξόνων 2G/6G
- ◇ έναν αισθητήρα θερμοκρασίας
- ◇ έναν αισθητήρα φωτός
- ◇ οχτώ πολύχρωμα LEDs
- ◇ έξι αναλογικές εισόδους
- ◇ ψηφιακές εισόδους και εξόδους
- ◇ δύο διακόπτες
- ◇ πέντε ακίδες γενικού σκοπού
- ◇ τέσσερις ακίδες υψηλού ρεύματος

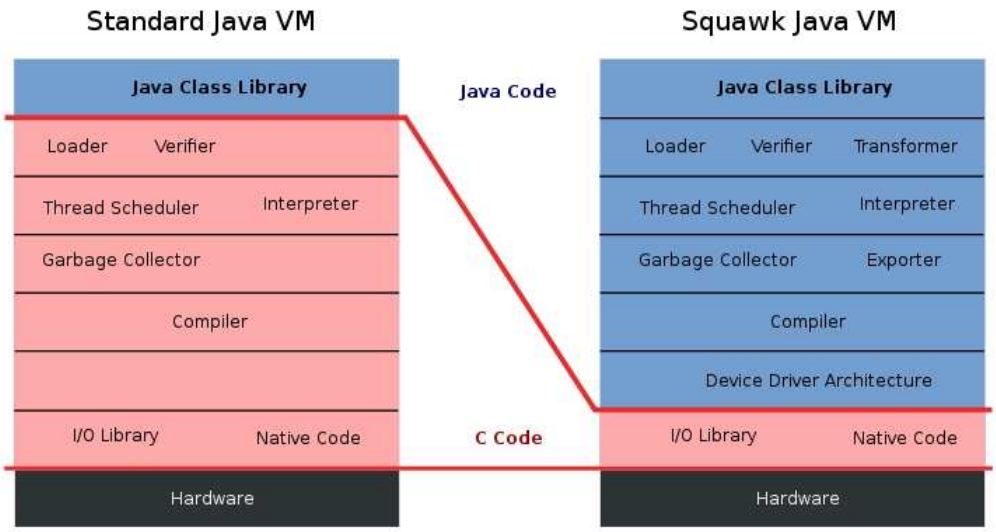
◇ μία επαναφορτιζόμενη μπαταρία

Από τα παραπάνω χαρακτηριστικά βλέπουμε ότι κάθε SunSPOT χρησιμοποιεί το πρωτόκολλο IEEE 802.15.4 για την ασύρματη επικοινωνία. Έτσι, ένα σύνολο από SunSPOTs μπορεί να αποτελέσει ένα ασύρματο δίκτυο αισθητήρων (Wireless Sensor Network - WSN) με το κάθε SunSPOT να αποτελεί έναν κόμβο του δικτύου.

6.1.2 Εικονική Μηχανή Squawk JVM

Η εικονική μηχανή Squawk JVM κατασκευάστηκε από την εταιρεία πληροφορικής Sun Microsystems για την ανάπτυξη εφαρμογών Java που προορίζονταν για ενσωματωμένα συστήματα και μικροσυσκευές, και αποτελεί το βασικό λογισμικό των SunSPOTs. Η κατασκευή αυτής της εικονικής μηχανής προέκυψε από την ανάγκη να δημιουργηθεί ένα μηχάνημα εικονικής πραγματικότητας σε Java, συμβατό με τη λειτουργία CLDC, (Connected Limited Device Configuration), που δε θα χρειαζόταν την ύπαρξη λειτουργικού συστήματος για να εκτελεστεί.

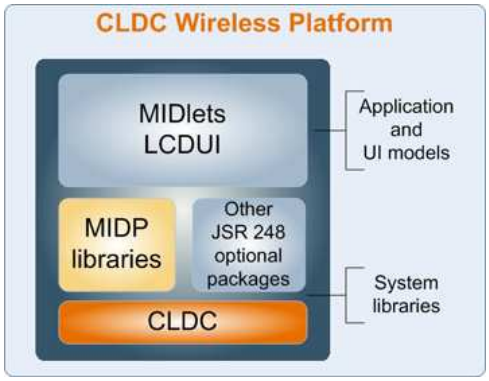
Έτσι, ο πύρηνας του Squawk JVM σε αντίθεση με τις περισσότερες εικονικές μηχανές που έχουν γραφτεί σε χαμηλού επιπέδου γλώσσες, όπως C/C++ ή assembly, έχει γραφτεί σχεδόν ολόκληρος σε Java. Το Squawk JVM, λοιπόν, δε χρειάζεται την ύπαρξη λειτουργικού συστήματος για να εκτελεστεί, ενώ παράλληλα προσφέρει πολλά από τα πλεονεκτήματα της κανονικής έκδοσης της Java SE, όπως είναι η συλλογή απορριμμάτων (garbage collector) και ο χειρισμός εξαιρέσεων (exception handling). Ταυτόχρονα απαιτεί μικρή μνήμη, διευκολύνει τη μεταγλώττιση (compiling) και την εκτέλεση του κώδικα και παρέχει τη δυνατότητα ταυτόχρονης εκτέλεσης πολλαπλών εφαρμογών, με την κάθε εφαρμογή να είναι πλήρως ανεξάρτητη και απομονωμένη από όλες τις υπόλοιπες. Το Squawk JVM αποτελεί λογισμικό ανοιχτού κώδικα και η αρχιτεκτονική του φαίνεται στην επόμενη εικόνα.



Σχήμα 16: *Αρχιτεκτονική δομή εικονικής μηχανής Squawk JVM*

6.1.3 MIDlets

Οι εφαρμογές που αναπτύσσονται για τα SunSPOTs ονομάζονται MIDlets. Τα MIDlets χρησιμοποιούν τις προδιαγραφές του MIDP, (Mobile Information Device Profile), όπως αυτές ορίζονται από το πρωτόκολλο CLDC, (Connected Limited Device Configuration), και προορίζονται για εφαρμογή σε περιβάλλον Java ME (Micro Edition), το οποίο είναι εγκατεστημένο στις περισσότερες ενσωματωμένες συσκευές.

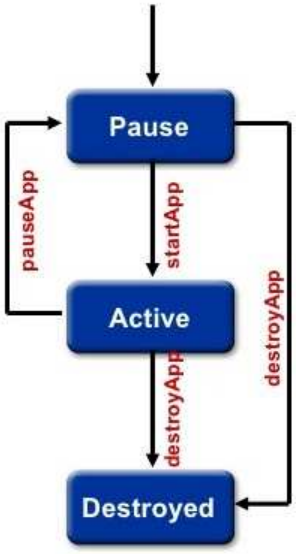


Το πρωτόκολλο CLDC περιγράφει το βασικό σύνολο των βιβλιοθηκών και των χαρακτηριστικών της εικονικής μηχανής, που θα πρέπει να υπάρχει σε μία πλατφόρμα (π.χ. κινητό, τάμπλετ, κ.λπ.), ώστε να είναι δύνατη η εφαρμογή προγραμμάτων Java. Το πρωτόκολλο CLDC συνδυάζεται με ένα ή περισσότερα προφίλ (π.χ. το MIDP) για να δώσει στους προγραμματιστές μια πλατφόρμα πάνω στην οποία μπορούν να δημιουργήσουν εφαρμογές

για ενσωματωμένες φορητές συσκευές με πολύ περιορισμένους πόρους.

Τυπικές εφαρμογές των MIDlets αποτελούν τα παιχνίδια σε κινητά τηλέφωνα, τα οποία υποστηρίζουν γραφικά, χρήση πληκτρολογίου για επικοινωνία με το χρήστη και περιορισμένη σύνδεση στο διαδίκτυο μέσω HTTP. Οι τρεις πιθανές καταστάσεις στον κύκλο ζωής ενός MIDlet (βλ.Σχήμα 17) είναι οι εξής:

1. **paused**: το MIDlet έχει δημιουργηθεί και είναι ανενεργό
2. **active**: το MIDlet είναι ενεργό
3. **destroyed**: το MIDlet έχει καταστραφεί και αναμένεται η αποκατάστασή του από τον συλλέκτη απορριμμάτων.



Σχήμα 17: *Κύκλος ζωής ενός MIDlet*

Στην κανονική έκδοση της Java (Java SE) κάθε εφαρμογή που αναπτύσσεται πρέπει να περιέχει πάντα μία μέθοδο main() για να μπορέσει να εκτελεστεί. Στην περίπτωση, όμως, της έκδοσης Java ME, που υλοποιεί η Squawk VM, κάθε εφαρμογή που υλοποιείται πρέπει να είναι συμβατή με το πρότυπο MIDlet. Όλες οι εφαρμογές για τα SPOTs πρέπει να κληρονομούν (extend) τα στοιχεία της κλάσης MIDlet και να υλοποιούν τις παρακάτω μεθόδους:

i) startApp() - Η μέθοδος αυτή καλείται όταν πρόκειται να εκτελεστεί το MIDlet.

ii) pauseApp() - Η μέθοδος αυτή καλείται όταν πρόκειται να ανασταλεί η εκτέλεση του MIDlet.

iii) destroyApp() - Η μέθοδος αυτή καλείται όταν το MIDlet τερματίζεται από το σύστημα, όπως συμβαίνει στην περιπτώση που το MIDlet προκαλέσει μια εξαίρεση του τύπου MIDletStateChangeException.

Το MIDlet από την κατασκευή του βρίσκεται στην κατάσταση αναστολής **paused**. Με την μέθοδο startApp() δεσμεύονται οι απαραίτητοι πόροι για την εκτέλεση της εφαρμογής και ενεργοποιείται το MIDlet. Στην περίπτωση που η εκτέλεση του MIDlet γίνει ανενεργή ή συμβεί μία εξαίρεση, τότε καλείται η μέθοδος pauseApp(), η οποία απελευθερώνει προσωρινά τους αρχικά δεσμευμένους πόρους. Μετά την ολοκλήρωση της εκτέλεση της εφαρμογής καλείται η destroyApp(), η οποία αποδεσμεύει πλήρως τους δεσμευμένους πόρους του MIDlet και στη συνέχεια το καταστρέφει (συλλέκτης απορριμμάτων).

Οι εφαρμογές, λοιπόν, που προορίζονται για τα SPOTs έχουν την παρακάτω βασική δομή, στην οποία έχουν εισαχθεί όλες οι βιβλιοθήκες που απαιτούνται για την πλήρη λειτουργικότητά τους.

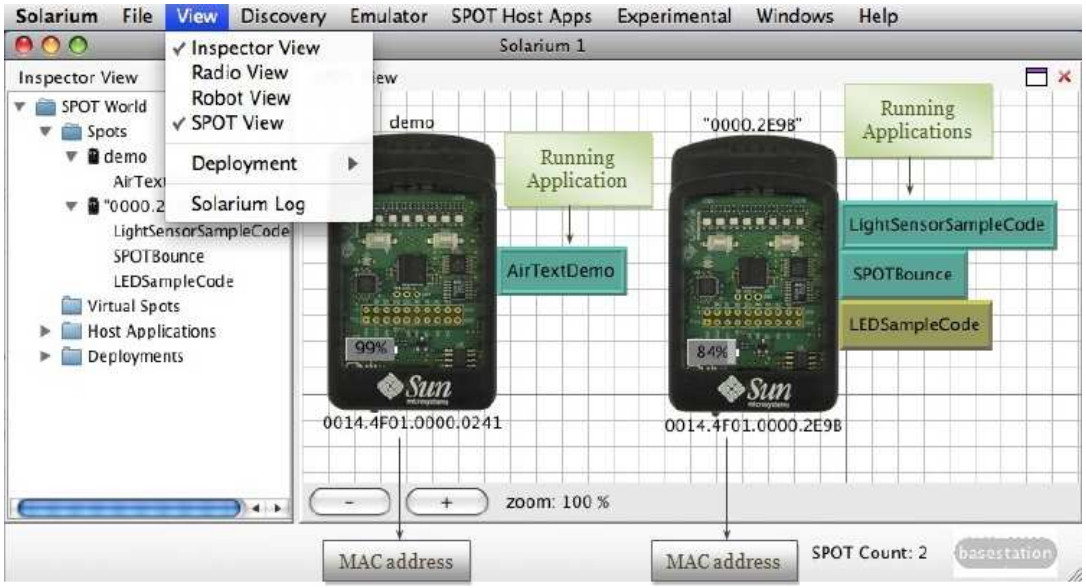
```
import com.sun.spot.peripheral.Spot;
import com.sun.spot.io.j2me.radiostream.*;
import com.sun.spot.io.j2me.radiogram.*;
import com.sun.spot.sensorboard.peripheral.ITriColorLED;
import com.sun.spot.sensorboard.EDemoBoard;
import com.sun.spot.peripheral.radio.IRadioPolicyManager;
import com.sun.spot.util.*;
import javax.microedition.io.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class SunSpotApplication extends MIDlet {
        protected void startApp() throws MIDletStateChangeException {
        }
        protected void pauseApp() {
                // This is not currently called by the Squawk VM
        }
        protected void destroyApp(boolean unconditional) throws
                MIDletStateChangeException {
        }
}
```

6.1.4 Εξομοιωτής Solarium

Το Solarium είναι μία εφαρμογή Java, η οποία εκτελείται στον υπολογιστή και χρησιμοποιείται για να εκτελέσει μια σειρά από ενέργειες στα SPOTs σαν εξομοιωτής (βλ.Σχήμα 18). Έχει τις παρακάτω δυνατότητες:

- Ανακαλύπτει και εμφανίζει SPOTs, τα οποία μπορεί να είναι συνδεδεμένα με τον υπολογιστή μέσω USB ή μέσω ασύρματης επικοινωνίας (Wireless).
- Μπορεί να φορτώσει και να ξεφορτώσει λογισμικό από τα SPOTs, να εκκινήσει, να διακόψει, να επανεκκινήσει και να τερματίσει εφαρμογές καθώς και να ανιχνεύσει την κατάσταση μιας συσκευής. Υπάρχει ένα πλαίσιο που λέγεται Radio View, το οποίο παρέχει έναν γραφικό τρόπο αναπαράστασης της ασύρματης συνδεσιμότητας όλων των SPOTs.
- Μπορεί να ελέγξει ένα δίκτυο από SPOTs με χρήση του πλαισίου Deployment View. Το πλαίσιο αυτό καθορίζει το ποιά εφαρμογή πρέπει να φορτωθεί σε ποιά συσκευή και διευκολύνει τη διαδικασία φόρτωσής τους.
- Περιλαμβάνει έναν προσομοιωτή, ο οποίος μπορεί να φορτώσει και να εκτελέσει εικονικές εφαρμογές σε εικονικά SPOTs (Virtual SPOTs).

Με τον εξομοιωτή αυτό δίνεται η δυνατότητα στον χρήστη να ελέγξει ένα πρόγραμμα που έχει αναπτύξει προτού το εφαρμόσει πάνω σε ένα πραγματικό SPOT. Ακόμα, όμως, και αν ένα πραγματικό SPOT δεν είναι διαθέσιμο, ο εξομοιωτής δίνει στο χρήστη τη δυνατότητα να χρησιμοποιήσει τα εικονικά SPOTs (virtal SunSPOTs). Ένα εικονικό SPOT περιλαμβάνει έναν εικονικό πίνακα αισθητήρων, όπου ο χρήστης μπορεί να θέσει τιμές στις διάφορες εικονικές παραμέτρους, (φωτισμού, θερμοκρασίας, κίνησης κτλ), καθώς και να ελέγξει τα LEDs, όπως ακριβώς και σε ένα πραγματικό SPOT. Το Solarium υποστηρίζει ακόμα και την αποστολή και τη λήψη δεδομένων μέσω της ασύρματης σύνδεσης για τα εικονικά SPOTs. Για την ασύρματη σύνδεση ορίζεται μία διεύθυνση σε κάθε εικονικό SPOT (MAC address) ώστε κάθε εικονικό SPOT να μεταδίδει και να λαμβάνει δεδομένα από άλλα εικονικά SPOTs.



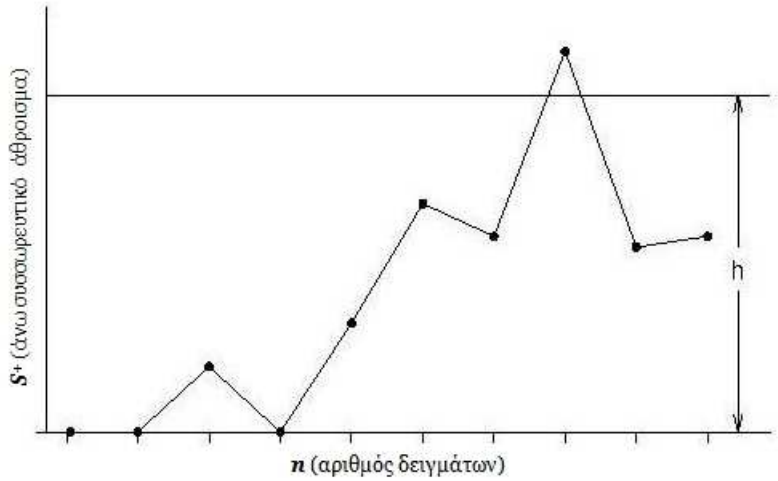
Σχήμα 18: *Εξομοιωτής Solarium*

6.2 Εφαρμογή Αλγορίθμου CUSUM για μία Μεταβλητή

Η εφαρμογή που αναπτύχθηκε για τον αλγόριθμο CUSUM μίας μεταβητής, έχει βασιστεί πάνω στον αλγόριθμο που περιγράφεται στην επόμενη παράγραφο (βλ.Αλγόριθμος 6.1). Ο αλγόριθμος αυτός είναι μονόπλευρος (one-sided CUSUM), που σημαίνει ότι υπολογίζει μόνο το άνω συσσωρευτικό άθροισμα S^+ και το συγκρίνει με το άνω όριο $+h$, όπως φαίνεται και στο διάγραμμα του Σχήματος 19.

6.2.1 Αλγόριθμος Μονόπλευρου CUSUM

Ο συγκεκριμένος αλγόριθμος για τον μονόπλευρο μονομεταβλητό CUSUM που αναπτύχθηκε ανιχνεύει τις αλλαγές που συμβαίνουν στην κατανομή μιας συνάρτησης x_t. Όπως αναφέρθηκε και προηγουμένως, ο αλγόριθμος αρχικά υπολογίζει το συσσωρευτικό άθροισμα, $R = S^+ = max(0, x_t - (\mu + k^+) + R)$ και στη συνέχεια το συγκρίνει με το επιλεγμένο άνω όριο h.



Σχήμα 19: *Διάγραμμα Μονόπλευρου CUSUM*

Τη στιγμή που μία τιμή του αθροίσματος R υπερβεί το όριο h ($R > h$), τότε ο αλγόριθμος θα δώσει σαν αποτέλεσμα 1, που σημαίνει ότι ανίχνευσε μία αλλαγή πάνω στην κατανομή της x_t, η οποία είναι εκτός ορίου h. Στην περίπτωση που το άθροισμα R πέσει κάτω από το όριο h, τότε ο αλγόριθμος για εκείνη τη συγκεκριμένη στιγμή θα μας δώσει την τιμή -1. Τέλος, αν μετά από κάποια αλλαγή ($R > h$ ή $R < h$) το άθροισμα R δεν αλλάξει κατάσταση, τότε ο αλγόριθμος θα δώσει την τιμή 0. Οι παράμετροι που χρησιμοποιούνται σαν είσοδοι στον αλγόριθμο μονόπλευρου CUSUM είναι οι εξής:

1. η μέση τιμή $\mu \in \mathbb{R}$,
2. το άνω όριο ανεκτικότητας k, και
3. το άνω όριο ελέγχου h

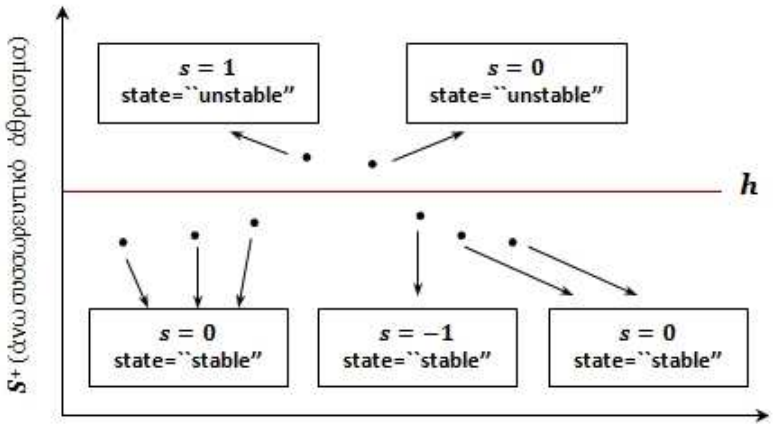
Ενώ σαν έξοδο μας δίνει ένα σήμα *output*, το οποίο έχουμε ορίσει να παίρνει μία εκ των τριών τιμών $output \in \{-1, 0, 1\}$, που αντιστοιχούν στις 3 καταστάσεις συστήματος που περιγράφηκαν παραπάνω, δηλαδή:

$output = 0$: Κατάσταση Σταθερή (είτε $R > h$ είτε $R < h$)

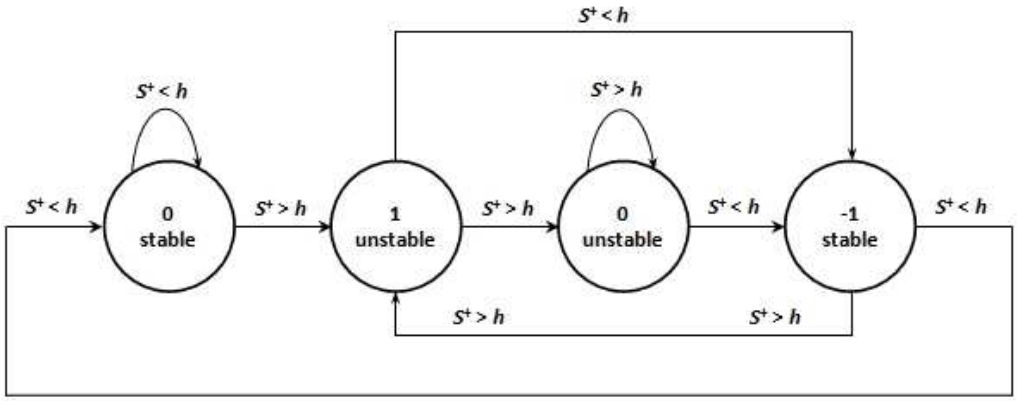
$output = 1$: Ανίχνευση αλλαγής πάνω από το όριο h τη χρονική στιγμή t_i

$output = -1$: Ανίχνευση αλλαγής κάτω από το όριο h τη χρονική στιγμή t_j

Επίσης, για το σωστό χαρακτηρισμό της κατάστασης στην οποία βρίσκεται το σύστημα που παρακολουθείται, έχει προστεθεί η παράμετρος *state*. Η παράμετρος αυτή κρατάει αποθηκευμένη (σαν μνήμη) την κατάσταση στην οποία βρίσκεται το σύστημα κάθε φορά που ελέγχει το συσσωρευτικό άθροισμα S^+. Έτσι, όταν το σύστημα είναι εντός ελέγχου η κατάστασή του χαρακτηρίζεται ως *"stable"*, ενώ στην αντίθετη περίπτωση χαρακτηρίζεται ως *"unstable"* (βλ. Σχήμα 20 και Σχήμα 21).



Σχήμα 20: *Καταστάσεις Αλγορίθμου CUSUM*



Σχήμα 21: *Μηχανή Καταστάσεων Συστήματος (fsm)*

Ακολουθεί η διατύπωση του αλγορίθμου CUSUM.

Αλγόριθμος 6.1. One-sided CUSUM Algorithm

$R \leftarrow 0$

output $\leftarrow 0$

state $\leftarrow$ "stable"

while(true)

$R = max(0, x_t - (\mu + k) + R)$

if $(R > h)$

if (state = "stable")

output = 1

state = "unstable"

else-if (state = "unstable")

output = 0

end-if

$R = 0$

end-if $(R > h)$

if $(R < h)$

if (state = "stable")

output = 0

else-if (state = "unstable")

output = -1

state = "stable"

end-if

end-if $(R < h)$

END.

■

6.2.2 Εφαρμογή Αλγορίθμου CUSUM στο SunSPOT

Η εφαρμογή του αλγορίθμου CUSUM που υλοποιήσαμε, αναπτύχθηκε σε γλώσσα Java με τη βοήθεια του αναπτυξιακού λογισμικού NetBeans 6.5 (IDE). Το NetBeans περιλαμβάνει μία πρόσθετη εφαρμογή (plug-in) που ονομάζεται SunSPOT Manager. Το SunSPOT Manager είναι ένα εργαλείο με το οποίο γίνεται δυνατή η διαχείριση πραγματικών SunSPOT, αλλά και εικονικών SunSPOT, μιας και περιλαμβάνει τον εξομοιωτή Solarium. Έτσι, με το συγκεκριμένο λογισμικό πακέτο μας δίνεται η δυνατότητα να ελέγξουμε την ορθότητα της εφαρμογής μας φορτώνοντάς την σε ένα εικονικό SunSPOT μέσω του εξομοιωτή Solarium.

Στην υλοποίησή μας, ο αλγόριθμος CUSUM εφαρμόζεται πάνω στις τιμές της θερμοκρασίας που περιλαμβάνονται μέσα σε ένα αρχείο. Στη συγκεκριμένη εφαρμογή, δηλαδή, δεν χρησιμοποιούνται οι τιμές που μας δίνει ο αισθητήρας θερμοκρασίας του SunSPOT.

Για να μπορέσει να είναι προσβάσιμο ένα αρχείο από μια εφαρμογή, κατά τη διάρκεια εκτέλεσής της σε ένα SunSPOT, θα πρέπει το αρχείο αυτό να είναι αποθηκευμένο στον φάκελο resources. Ο φάκελος αυτός δημιουργείται κατά τη δημιουργία ενός νέου πρότζεκτ στο NetBeans και βρίσκεται μέσα στον κύριο φάκελο του πρότζεκτ αυτού. Όλα τα αρχεία που βρίσκονται μέσα στο φάκελο resources είναι διαθέσιμα για κάθε εφαρμογή που εκτελείται σε ένα SunSPOT.

Για το διάβασμα του αρχείου temperature.txt, που περιλαμβάνει τις μετρήσεις της θερμοκρασίας με τις οποίες θα προμηθεύσουμε και θα ελέγξουμε τον αλγόριθμο CUSUM, φορτώθηκαν οι ακόλουθες κλάσεις:

- **Inputstream**: Η κλάση αυτή είναι αφηρημένη (abstract class) και υπερκλάση (uper-class) όλων των κλάσεων που εκπροσωπούν ρεύματα εισόδου από bytes.
- **InputStreamReader**: Η κλάση αυτή είναι ουσιαστικά η γέφυρα μεταξύ ενός ρεύματος εισόδου από bytes και ενός ρεύματος χαρακτήρων. Δηλαδή, η κλάση αυτή διαβάζει τα bytes από ένα ρεύμα εισόδου και τα αποκωδικοποιεί σε χαρακτήρες,

χρησιμοποιώντας ένα συγκεκριμένο σύνολο χαρακτήρων.

- **BufferedReader**: Η κλάση αυτή διαβάζει το κείμενο από μια ροή χαρακτήρων εισόδου, και με τη χρήση μιας ενδιάμεσης μνήμης (buffer) αποθηκεύει τους χαρακτήρες, εξασφαλίζοντας έτσι την αποτελεσματική ανάγνωση λέξεων, πινάκων και γραμμών. Επίσης, η κλάση αυτή περιλαμβάνει τη μέθοδο readline() με την οποία μπορεί να διαβαστεί ένα αρχείο ανά γραμμή.

Έτσι, για το διάβασμα του αρχείου ο κώδικας διαμορφώνεται ως εξής:

```
import java.io.InputStream;
import java.io.InputStreamReader;
import com.sun.squawk.io.BufferedReader;

InputStream inputStream = getClass().getResourceAsStream("/temperature.txt");
InputStreamReader dataReader = new InputStreamReader(inputStream);
BufferedReader bufferedReader = new BufferedReader(dataReader);
String thisLine = null;
StringBuffer data = new StringBuffer();
double temperature;

System.out.flush();
try {
        while ((thisLine = bufferedReader.readLine()) != null) {
                data.append(thisLine + "\n");
                temperature = Double.parseDouble(thisLine);
                System.out.println("Data = " + temperature);
        }
} catch(Exception e) {
        e.printStackTrace();
}
```

Η κύρια κλάση της εφαρμογής μας **CusumApplication** περιλαμβάνει τις ακόλουθες κλάσεις:

- **CSState**: Η κλάση αυτή ουσιαστικά χρησιμοποιείται για να υπάρχει αποθηκευμένη η ολική εικόνα του συστήματος (συσσωρευτικό άθροισμα και κατάσταση ελέγχου), προτού τρέξει ξανά ο αλγόριθμος CUSUM. Στην κλάση αυτή ορίζονται δύο μεταβλητές, ένας πρότυπος κατασκευαστής (default constructor) και δύο μέθοδοι. Στη συνέχεια, δίνεται μια σύντομη περιγραφή των μεταβλητών και των μεθόδων.
  - Η μεταβλητή **private double R**: κρατάει αποθηκευμένη την πιο πρόσφατη τιμή του συσσωρευτικού αθροίσματος S^+,

  - Η μεταβλητή **String csmstate**: κρατάει αποθηκευμένη την κατάσταση του συστήματος και μπορεί να είναι ίση μόνο με δύο συγκεκριμένες γραμματοσειρές. Αυτές οι γραμματοσειρές ορίζονται από την εσωτερική κλάση **MyEnum** και είναι η "STABLE" και η "UNSTABLE".
  - Η μέθοδος **public double getR()**: χρησιμοποιείται για το διάβασμα της τιμής της μεταβλητής R, εφόσον είναι κρυφή (private).
  - Η μέθοδος **public void setR(double R)**: θέτει μια καινούρια τιμή στην κρυφή μεταβλητή R.

- **CSAlgorithm**: Η κλάση αυτή είναι ο ίδιος ο αλγόριθμος CUSUM, όπως έχει περιγραφεί στην παράγραφο 6.2.1. Οι μεταβλητές που ορίζονται στην κλάση αυτή καθώς και οι μέθοδοι περιγράφονται παρακάτω.

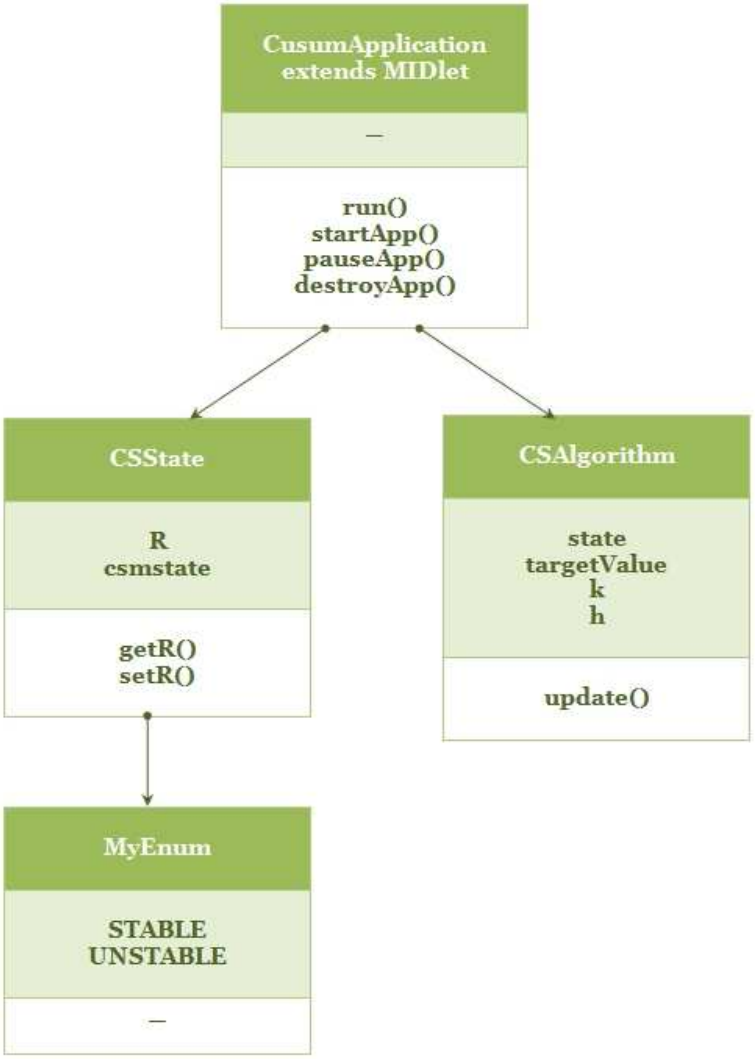
  - Η μεταβλητή **private CSState state**: ορίστηκε για να δίνει τις απαραίτητες πληροφορίες στον αλγόριθμο όσον αφορά την προηγούμενη κατάσταση του συστήματος.
  - Η μεταβλητή **double targetValue**: περιέχει τη μέση τιμή της θερμοκρασίας.
  - Η μεταβλητή **double k**: έχει οριστεί ως το άνω όριο της ανεκτικότητας.
  - Η μεταβλητή **double h**: έχει οριστεί ως το άνω όριο ελέγχου για τον αλγόριθμο.
  - Η μέθοδος **public int update(double xt)**: υπολογίζει το καινούριο άνω συσσωρευτικό άθροισμα S^+ και αφού το συγκρίνει με το άνω όριο h μας επιστρέφει την τελική έξοδο του αλγορίθμου **output** $\in \{-1, 0, 1\}$.

Επίσης, η κύρια κλάση **CusumApplication** εκτός από τις τρεις μεθόδους που ορίζονται πάντα για τη σωστή λειτουργία του MIDlet και άρα και του SPOT (startApp(), pauseApp(), destroyApp()), περιλαμβάνει ακόμη την:

- **private void run()**: η οποία τεστάρει την εφαρμογή μέσω του διαβάσματος των

τιμών της θερμοκρασίας, που περιέχονται μέσα στο αρχείο temperature.txt και μας δίνει τα ανάλογα αποτελέσματα.

Στο σχήμα 22 δίνεται ένα διάγραμμα με τις κλάσεις της εφαρμογής CusumApplication για μια καλύτερη εικόνα του τρόπου με τον οποίο συνδέονται μεταξύ τους. Έπειτα ακολουθεί ολόκληρος ο κώδικας σε Java, όπως αναπτύχθηκε για τον αλγορίθμο μονόπλευρου CUSUM για μία μεταβλητή, καθώς και τα αποτελέσματα που πήραμε από τον εξομοιωτή κατά την εφαρμογή του πάνω σε ένα εικονικό SPOT (βλ.Σχήμα 23).

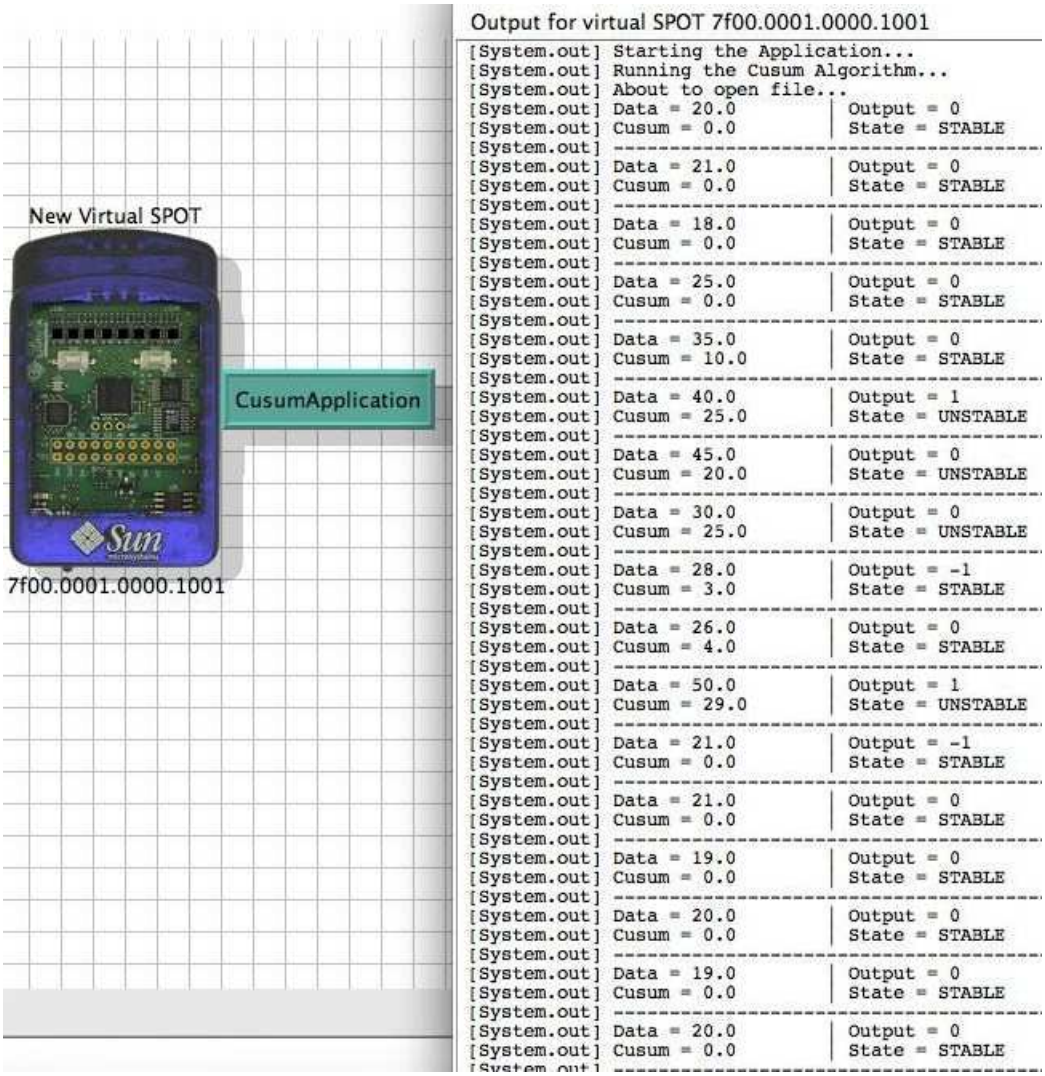


Σχήμα 22: *Κλάσεις της εφαρμογής CusumApplication*

```
package net.java.dev.netbeansspot;

// Classes needed for the MIDlet
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

// Class for Exception messages
import java.io.IOException;

// Classes for Reading a File
import java.io.InputStream;
import java.io.InputStreamReader;
import com.sun.squawk.io.BufferedReader;

public class CusumApplication extends MIDlet {
  // Class to define the system's state
  public class CSState {
    private double R;
    String csmstate;

    /* Two states: STABLE, UNSTABLE                    */
    /* STABLE    ---> Temperature under the limit      */
    /* UNSTABLE  ---> Temperature over the limit       */

    public class MyEnum {
      public final static String STABLE = "STABLE";
      public final static String UNSTABLE = "UNSTABLE";
    }

    // Default constructor
    public CSState() {
      R = 0;
      csmstate = "STABLE";
    }

    // Method for getting the value of R
    public double getR() {
      return R;
    }

    // Method for setting a new value to R
    public void setR(double R) {
      this.R = R;
    }
  } // end of class CSState
```

```
public class CSAlgorithm {
  private CSState state = null;
  double targetValue;   // parameter
  double k;         // above tolerance
  double h;         // above threshold

  // Default Constructor
  public CSAlgorithm(){ }

  // Constructor with parameters
  public CSAlgorithm(double TargetValue, double kPlus, double hPlus) {
    state = new CSState();
    this.targetValue = TargetValue;
    this.k = kPlus;
    this.h = hPlus;
  }

  // Method that defines the final output of the algorithm
  public int update(double xt) {
    int outValue = 0;
    double RR;

    state.setR(Math.max(0, (xt - (targetValue + k) + state.getR())));
    RR = state.getR();

    if (RR < h) {
      if (state.csmstate.equalsIgnoreCase(CSState.MyEnum.STABLE)) {
        outValue = 0;
      }
      else if (state.csmstate.equalsIgnoreCase(CSState.MyEnum.UNSTABLE)) {
        state.csmstate = CSState.MyEnum.STABLE;
        outValue = -1;
      }
    }
    else if (RR > h) {
      if (state.csmstate.equalsIgnoreCase(CSState.MyEnum.STABLE)) {
        state.csmstate = CSState.MyEnum.UNSTABLE;
        outValue = 1;
      }
      else if (state.csmstate.equalsIgnoreCase(CSState.MyEnum.UNSTABLE)) {
        outValue = 0;
      }
      state.setR(0.0);
    }
    return outValue;
  } // end of method update()
} // end of CSAlgorithm
```

```
  private void run() throws IOException {
    int output = 0;       // the final output of the algorithm
    double temperature;   // to save the new temperature from file

    // Construct a cusum with parameters
    // targetValue = 20.0, k = 5.0, h = 20.0
    CSAlgorithm cum = new CSAlgorithm(20.0, 5.0, 20.0);

    System.out.println("Running the Cusum Algorithm...");
    System.out.println("About to open file...");

    // We define the necessary variables for reading a file
    InputStream inputStream = getClass().getResourceAsStream("/temperature.txt");
    InputStreamReader dataReader = new InputStreamReader(inputStream);
    BufferedReader bufferedReader = new BufferedReader(dataReader);
    String thisLine = null;
    StringBuffer data = new StringBuffer();

    // Flush the input stream from leftover characters
    System.out.flush();

    try {
      // Loop until the file ends
      while ((thisLine = bufferedReader.readLine()) != null) {
        data.append(thisLine + "\n");
        temperature = Double.parseDouble(thisLine);
        output = cum.update(temperature);
        System.out.println("Data = "+temperature+"\t| Output = "+output);
        System.out.println("Cusum = "+cum.RR+"\t| State = "+cum.state.csmstate);
        System.out.println("-----------------------------------------");
      }
    } catch(Exception e) {
      e.printStackTrace();
    } // end of try
  } // end of method run()

  protected void startApp() throws MIDletStateChangeException {
    System.out.println("Starting the Application...");
    try {
      run();
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }

  protected void pauseApp() {
    // This is not currently called by the Squawk VM
  }

  protected void destroyApp(boolean unconditional) throws
     MIDletStateChangeException {
    // TODO
  }
} // end of class CusumApplication
```



Σχήμα 23: *Αποτελέσματα εφαρμογής αλγορίθμου CUSUM*

6.3 Εφαρμογή Αλγορίθμου CUSUM για δύο Μεταβλητές

Η εφαρμογή που αναπτύχθηκε για τον αλγόριθμο CUSUM για δύο μεταβλητές έχει βασιστεί πάνω στον αλγόριθμο που περιγράφεται στην αμέσως επόμενη παράγραφο (βλ.Αλγό-ριθμος 6.2). Για τον αλγόριθμο αυτό χρησιμοποιούνται οι σχέσεις 5.3.4, 5.3.5 και 5.3.6, που περιγράφουν τον μονόπλευρο πολυμεταβλητό αλγόριθμο CUSUM, σύμφωνα με τον Crosier (1988). Για το λόγο αυτό, επιλέξαμε να ονομάσουμε τον συγκεκριμένο αλγόριθμο ως *αλγόριθμο Crosier*.

6.3.1 Αλγόριθμος Crosier

Στη λογική του ο αλγόριθμος Crosier δεν διαφέρει από τον αλγόριθμο 6.1, που περιγράφει τον αλγόριθμο μονόπλευρου CUSUM. Στο μόνο που διαφέρουν είναι στον τρόπο που θα υπολογιστεί το τελικό άνω συσσωρευτικό άθροισμα. Σαν είσοδοι στον αλγόριθμο Crosier έχουν οριστεί:

1. το διάνυσμα των μέσων τιμών **means**,
2. ο αντίστροφος πίνακας συνδιακύμανσης $\mathbf{\Sigma}^{-1}$,
3. το άνω όριο ανεκτικότητας k και
4. το άνω όριο ελέγχου h.

Η έξοδος του αλγορίθμου είναι η μεταβλητή *output*, η οποία όπως και στον αλγόριθμο CUSUM έχει οριστεί ώστε να παίρνει μία εκ των τριών τιμών $\{-1, 0, 1\}$, δηλαδή, $output \in \{-1, 0, 1\}$. Επίσης, η μεταβλητή *state* που είχαμε ορίσει στον αλγόριθμο CUSUM χρησιμοποιείται και στον αλγόριθμο Crosier ακριβώς με την ίδια λογική και οι καταστάσεις του συστήματος όπως περιγράφονται από τα σχήματα 20 και 21 ισχύουν και σε αυτή την περίπτωση. Ακολουθεί η διατύπωση για τον αλγόριθμο Crosier.

Αλγόριθμος 6.2. Crosier Algorithm

$Y_n \leftarrow 0$

ouput $\leftarrow 0$

state $\leftarrow$ "stable"

while(true)

$C_n = \left[(\mathbf{S}_{n-1} + \mathbf{x}_n - \mathbf{means})'\mathbf{\Sigma}^{-1}(\mathbf{S}_{n-1} + \mathbf{x}_n - \mathbf{means})\right]^{1/2}$

if $(C_n \geq k)$

$\mathbf{S}_n = (1 - k/C_n)(\mathbf{S}_{n-1} + \mathbf{x}_n - \boldsymbol{\mu})$

else-if $(C_n < k)$

$\mathbf{S}_n = 0$

end-if

$Y_n = [\mathbf{S}_n'\mathbf{\Sigma}^{-1}\mathbf{S}_n]^{1/2}$

if $(Y_n > h)$

if (state = "stable")

output = 1

state = "unstable"

else-if (state = "unstable")

output = 0

end-if

end-if $(Y_n > h)$

if $(Y_n < h)$

if (state = "stable")

output = 0

else-if (state = "unstable")

ouput $= -1$

state = "stable"

end-if

end-if $(Y_n < h)$

END.

■

6.3.2 Εφαρμογή Αλγορίθμου Crosier στο SunSPOT

Η κύρια κλάση της εφαρμογής μας **CrosierApplication** περιλαμβάνει τις ακόλουθες κλάσεις:

- **CrosierState**: Η κλάση αυτή ουσιαστικά χρησιμοποιείται για να υπάρχει αποθηκευμένη η ολική εικόνα του συστήματος (συσσωρευτικό άθροισμα και κατάσταση ελέγχου), προτού τρέξει ξανά ο αλγόριθμος Crosier. Στην κλάση αυτή ορίζο-νται δύο μεταβλητές, ένας πρότυπος κατασκευαστής (default constructor) και δύο μέθοδοι. Στη συνέχεια, δίνεται μια σύντομη περιγραφή των μεταβλητών και των μεθόδων.
  - Η μεταβλητή **private double Y**: κρατάει αποθηκευμένη την πιο πρόσφατη τιμή του συσσωρευτικού αθροίσματος, όπως αυτό υπολογίζεται από τη σχέση 5.3.6.
  - Η μεταβλητή **String crosierstate**: κρατάει αποθηκευμένη την κατάσταση του συστήματος και μπορεί να είναι ίση μόνο με δύο συγκεκριμένες γραμματοσειρές. Αυτές οι γραμματοσειρές ορίζονται από την εσωτερική κλάση **MyEnum** και είναι η "STABLE" και η "UNSTABLE".
  - Η μέθοδος **public double getY()**: χρησιμοποιείται για το διάβασμα της τιμής της μεταβλητής Y, εφόσον είναι κρυφή (private).
  - Η μέθοδος **public void setY(double Y)**: θέτει μια καινούρια τιμή στην κρυφή μεταβλητή Y.

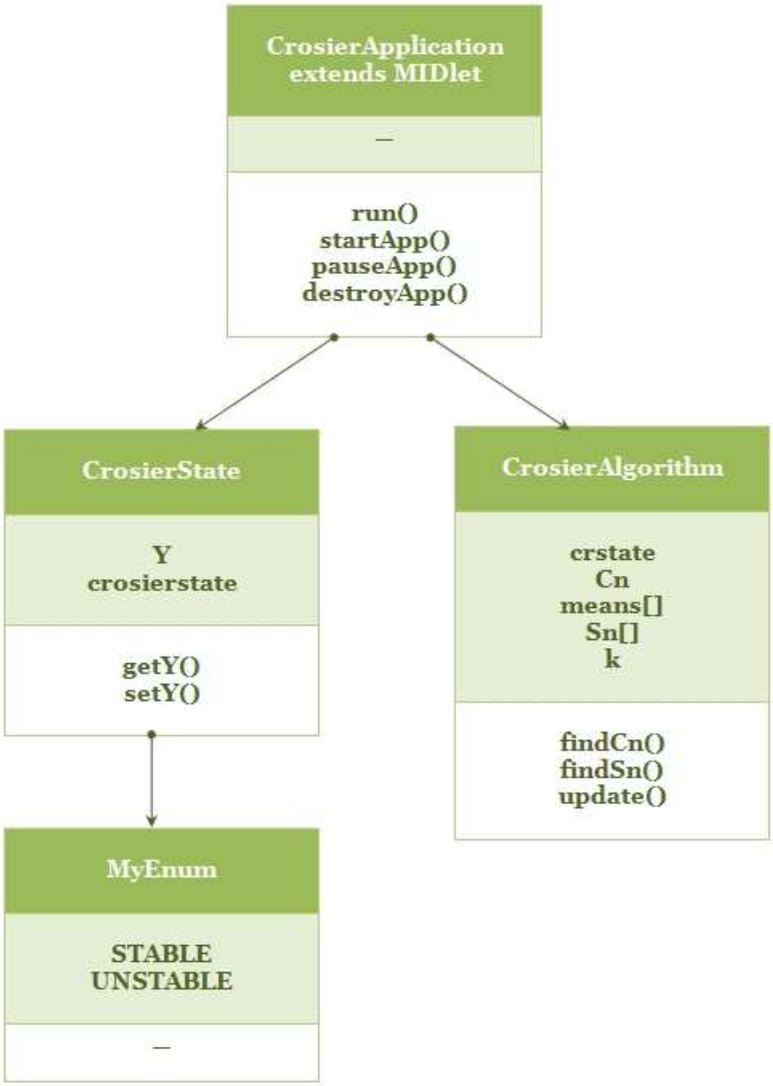
- **CrosierAlgorithm**: Η κλάση αυτή είναι ο ίδιος ο αλγόριθμος Crosier, όπως έχει περιγραφεί στην παράγραφο 6.3.1. Οι μεταβλητές που ορίζονται στην κλάση αυτή καθώς και οι μέθοδοι περιγράφονται παρακάτω.
  - Η μεταβλητή **private CrosierState crstate**: ορίστηκε για να δίνει τις απαραίτητες πληροφορίες στον αλγόριθμο όσον αφορά την προηγούμενη κατάσταση του συστήματος.
  - Η μεταβλητή **double Cn**: η οποία περιγράφεται από τη σχέση 5.3.4
  - Η μεταβλητή **double[] means**: είναι ο πίνακας που περιέχει τις μέσες τιμές των δύο μεταβλητών του αλγορίθμου. Αντιπροσωπεύει, δηλαδή, το διάνυσμα των μέσων τιμών.
  - Η μεταβλητή **double[] Sn**: είναι ο πίνακας (διάνυσμα), ο οποίος περιέχει τις τιμές που υπολογίζονται σύμφωνα με τη σχέση 5.3.5 που ορίζει το διάνυσμα $\mathbf{S_n}$.
  - Η μεταβλητή **double k**: έχει οριστεί ως το άνω όριο της ανεκτικότητας για το σύστημα.
  - Η μέθοδος **public int update(double[] covMat, double threshold)**: υπολογίζει το καινούριο άνω συσσωρευτικό άθροισμα Y και αφού το συγκρίνει με το άνω όριο h μας επιστρέφει την τελική έξοδο του αλγορίθμου **output** $\in \{-1, 0, 1\}$.

Επίσης, η κύρια κλάση **CrosierApplication** εκτός από τις τρεις μεθόδους που ορίζονται πάντα για τη σωστή λειτουργία του MIDlet και άρα και του SPOT (startApp(), pauseApp(), destroyApp()), περιλαμβάνει ακόμη τη μέθοδο:

- **private void run()**: η οποία ελέγχει την εφαρμογή μέσω του διαβάσματος τιμών δύο μεταβλητών x_1 και x_2, οι οποίες περιέχονται μέσα στα αρχεία input1.txt και input2.txt αντίστοιχα για την καθεμία. Επειδή, ο αλγόριθμος Crosier χρησιμοποιεί περίπλοκες σχέσεις και σύνθετους υπολογισμούς, για τον έλεγχο της ορθότητάς

του επιλέξαμε να χρησιμοποιήσουμε τις τιμές του παραδείγματος 5.1.

Στο σχήμα 24 δίνεται ένα διάγραμμα με τις κλάσεις της εφαρμογής CrosierApplication για μια καλύτερη εικόνα του τρόπου με τον οποίο συνδέονται μεταξύ τους. Έπειτα, ακολουθεί ολόκληρος ο κώδικας σε Java, όπως αναπτύχθηκε για τον αλγορίθμο Crosier για δύο μεταβλητές, καθώς και τα αποτελέσματα που πήραμε από τον εξομοιωτή κατά την εφαρμογή του πάνω σε ένα εικονικό SPOT (βλ.Σχήμα 25). Αν συγκρίνουμε τα αποτελέσματα της εφαρμογής μας με αυτά του παραδείγματος 5.1, τα οποία συνοψίζονται στον πίνακα 2, θα δούμε ότι είναι ίδια. Γεγονός, που συνεπάγεται ότι η εφαρμογή μας δίνει σωστά αποτελέσματα.



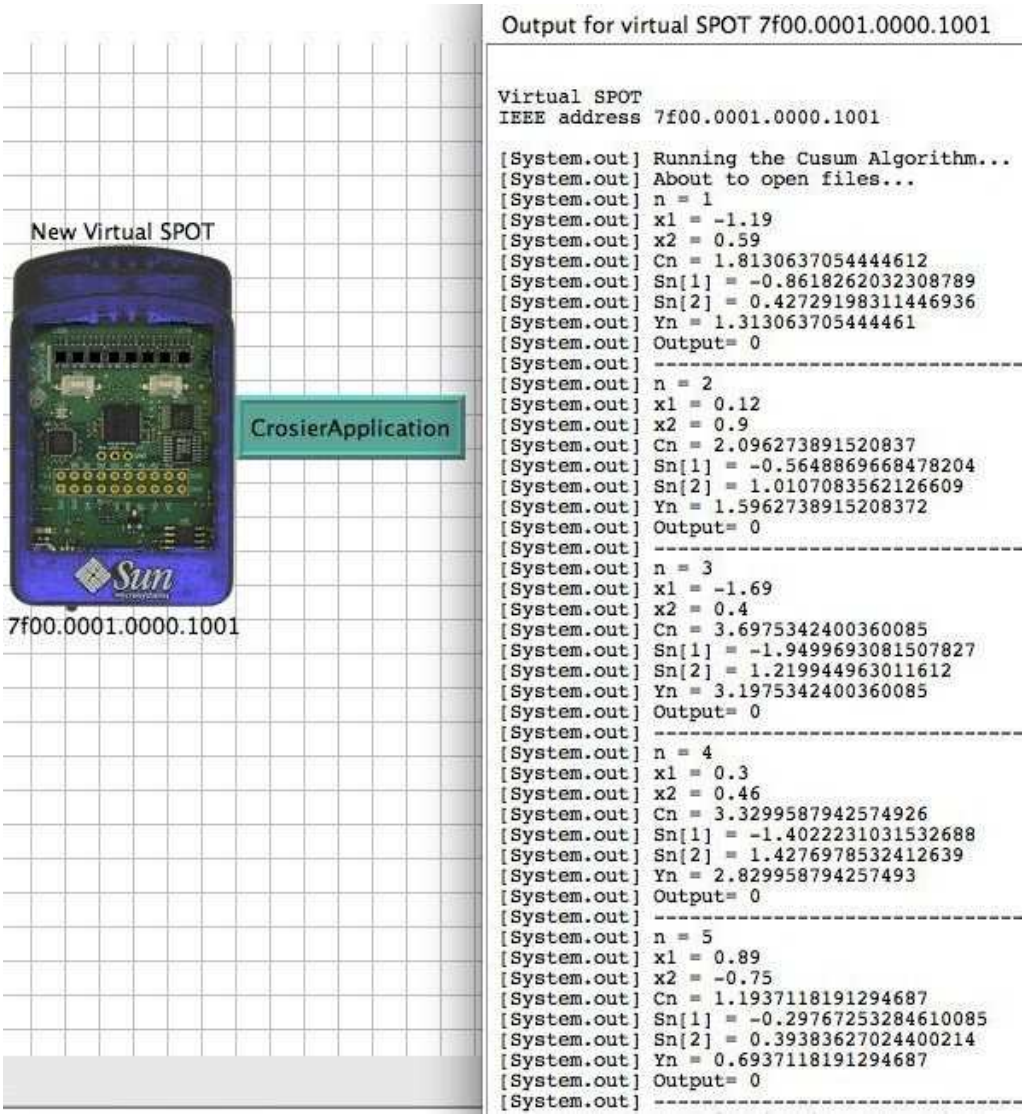
Σχήμα 24: *Κλάσεις της εφαρμογής CrosierApplication*

```
package net.java.dev.netbeansspot;

// Classes needed for the MIDlet
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

// Class for Exception messages
import java.io.IOException;

// Classes for Reading a File
import java.io.InputStream;
import java.io.InputStreamReader;
import com.sun.squawk.io.BufferedReader;

public class CrosierApplication extends MIDlet {
  // Class to define the system's state
  public class CrosierState {
    private double Y;
    String crosierstate;

    /* TWO states: STABLE, UNSTABLE                    */
    /* STABLE    ---> Temperature under the limit      */
    /* UNSTABLE  ---> Temperature over the limit       */

    public class MyEnum {
      public final static String STABLE = "STABLE";
      public final static String UNSTABLE = "UNSTABLE";
    }

    // Default constructor
    public CrosierState() {
      this.Y = 0.0;
      crosierstate = "STABLE";
    }

    // Method for getting the value of Y
    public double getY() {
      return Y;
    }

    // Method for setting a new value to Y
    public void setY(double Y) {
      this.Y = Y;
    }
  } // end of class CrosierState

  // Class for the CrosierAlgorithm
  public class CrosierAlgorithm {
    private CrosierState crstate = null;
    private double Cn;
    double[] means;
    double[] Sn;
    double k;

    // Default constructor
    public CrosierAlgorithm() {
      crstate = new CrosierState();
      this.Cn = 0.0;
    }
```

```
public CrosierAlgorithm(double k, double[] meanMatrix, double[] sn) {
  this.means = meanMatrix;
  this.Sn = sn;
  this.k = k;
  crstate = new CrosierState();
}
public CrosierAlgorithm(double k) {
  this.k = k;
  crstate = new CrosierState();
}

// Method for setting a new value to Cn
public void setCn(double cn) {
  this.Cn = cn;
}

// Method for reading the value of Cn
public double getCn() {
  return this.Cn;
}

// Method for finding the new value of Cn
public void findCn(double[] xn, double[] covMat) {
  double c = 0.0;
  double[] temp_cn = new double[2];
  double[] temp_mult = new double[2];

  temp_cn[0] = Sn[0] + xn[0] - means[0];
  temp_cn[1] = Sn[1] + xn[1] - means[1];
  temp_mult[0] = temp_cn[0] * covMat[0] + temp_cn[1] * covMat[2];
  temp_mult[1] = temp_cn[0] * covMat[1] + temp_cn[1] * covMat[3];
  c = temp_mult[0] * temp_cn[0] + temp_mult[1] * temp_cn[1];
  setCn(Math.sqrt(c));

  System.out.println("Cn = " + getCn());

  // Call method findSn to find the new vector Sn
  findSn(getCn(), xn);
} // end of method findCn

// Method for finding the new vector Sn
public void findSn(double cn, double[] xn) {
  double c = 0.0;
  double[] temp_sn = new double[2];

  c = 1 - k/getCn();

  temp_sn[0] = c * (Sn[0] + xn[0] - means[0]);
  temp_sn[1] = c * (Sn[1] + xn[1] - means[1]);

  Sn = temp_sn;

  System.out.println("Sn[1] = "+Sn[0]);
  System.out.println("Sn[2] = "+Sn[1]);
} // end of method findSn
```

```
    public int update(double[] covMat, double threshold) {
      int output = 0;
      double yn = 0.0;
      double[] temp_yn = new double[2];

      temp_yn[0] = Sn[0] * covMat[0] + Sn[1] * covMat[2];
      temp_yn[1] = Sn[0] * covMat[1] + Sn[1] * covMat[3];
      yn = Math.sqrt(temp_yn[0] * Sn[0] + temp_yn[1] * Sn[1]);

      System.out.println("Yn = "+yn);

      if (yn < threshold) {
        if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.STABLE)) {
          output = 0;
        }
        else if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.UNSTABLE)) {
          crstate.crosierstate = CrosierState.MyEnum.STABLE;
          output = -1;
        }
      } else {
        if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.STABLE)) {
          crstate.crosierstate = CrosierState.MyEnum.UNSTABLE;
          output = 1;
        }
        else if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.UNSTABLE)) {
          output = 0;
        }
      }
      return output;
    } // end of method update()
  } // end of class CrosierAlgorithm

  private void run() throws IOException {
    CrosierAlgorithm crosier = new CrosierAlgorithm(0.5);
    int count = 0;
    int outValue = 0;

    double[] means = {0.0, 0.0};
    double[] s0 = {0.0, 0.0};
    double[] x = new double[2];
    double[] covMatrix = {1.33, -0.67, -0.67, 1.33};

    System.out.println("Running the Cusum Algorithm ...");
    System.out.println("About to open files ...");

    // Read from file input1.txt
    InputStream inputStream1 = getClass().getResourceAsStream("/input1.txt");
    InputStreamReader dataReader1 = new InputStreamReader(inputStream1);
    BufferedReader buffReader1 = new BufferedReader(dataReader1);
    String thisLine1 = null;
    StringBuffer data1 = new StringBuffer();

    // Read from file input2.txt
    InputStream inputStream2 = getClass().getResourceAsStream("/input2.txt");
    InputStreamReader dataReader2 = new InputStreamReader(inputStream2);
    BufferedReader buffReader2 = new BufferedReader(dataReader2);
    String thisLine2 = null;
    StringBuffer data2 = new StringBuffer();
```

```
    System.out.flush();

    try {
      // Loop until the end of files
      while (((thisLine1 = buffReader1.readLine())!= null) &&
            ((thisLine2 = buffReader2.readLine())!= null))
      {
        data1.append(thisLine1 + "\n");
        x[0] = Double.parseDouble(thisLine1);
        data2.append(thisLine1 + "\n");
        x[1] = Double.parseDouble(thisLine2);

        System.out.println("n = "+(count+1));
        System.out.println("x1 = " + x[0]);
        System.out.println("x2 = " + x[1]);

        if (count == 0) {
          crosier.Sn = s0;
        }
        crosier.means = means;
        crosier.findCn(x, covMatrix);
        outValue = crosier.update(covMatrix, 5.5);
        System.out.println("Output= " + outValue);
        System.out.println("----------------------------------------");
        count++;
      } // end of while statement
    } catch(Exception e) {
          e.printStackTrace();
    } // end of try
  } // end of method run()

  protected void startApp() throws MIDletStateChangeException {
    try {
      run();
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }

  protected void pauseApp() {
    // This is not currently called by the Squawk VM
  }

  protected void destroyApp(boolean unconditional) throws
      MIDletStateChangeException {
    // TODO
  }
}
```



Σχήμα 25: *Αποτελέσματα εφαρμογής αλγορίθμου Crosier*

6.3.3 Εφαρμογή Αλγορίθμου Crosier σε δίκτυο από SunSPOTs

Το δίκτυο που υλοποίησαμε αποτελείται από τρία SunSPOTs. Το κάθε SunSPOT τρέχει τη δική του εφαρμογή, που είναι ειδικά διαμορφωμένη για τη λειτουργία που θέλουμε να εκτελέσει. Ανάμεσά τους υπάρχει επικοινωνία και είναι δυνατή η λήψη και η αποστολή μηνυμάτων και από τα τρία. Τα δύο SunSPOTs διαβάζουν δεδομένα από ένα αρχείο και τα στέλνουν στο τρίτο SunSPOT, το οποίο τα χρησιμοποιεί για να τρέξει τον αλγόριθμο Crosier. Το αρχείο από το οποίο διαβάζουν τα δύο SunSPOTs είναι διαφορετικό για το καθένα, μιας και το ένα περιλαμβάνει τιμές θερμοκρασίας και το άλλο τιμές φωτεινότητας.

Για το συγχρονισμό των τριών SunSPOTs, έχουμε ορίσει το SunSPOT που θα τρέξει τον αλγόριθμο Crosier να στείλει πρώτα ένα μήνυμα "OK" στα άλλα δύο, ώστε να αρχίσουν να στέλνουν τα δεδομένα τους. Επίσης, έχουμε ορίσει τα δύο SunSPOTs να μην ξεκινήσουν να στέλνουν δεδομένα προτού λάβουν το μήνυμα "OK" από το τρίτο. Τις τρεις εφαρμογές που αναπτύξαμε τις έχουμε αντιστοιχίσει σε συγκεκριμένα SunSPOTs ανάλογα με τη διεύθυνση που έχει το καθένα (MAC address). Συγκεκριμένα,

- ◇ το SunSPOT με τη διεύθυνση 7f00.0001.0000.1001 τρέχει την εφαρμογή **SenderTemperature**,
- ◇ το SunSPOT με τη διεύθυνση 7f00.0001.0000.1002 τρέχει την εφαρμογή **SenderBrightness** και τέλος,
- ◇ το SunSPOT με τη διεύθυνση 7f00.0001.0000.1003 τρέχει την εφαρμογή **CrosierOnlineAppReceiver**.

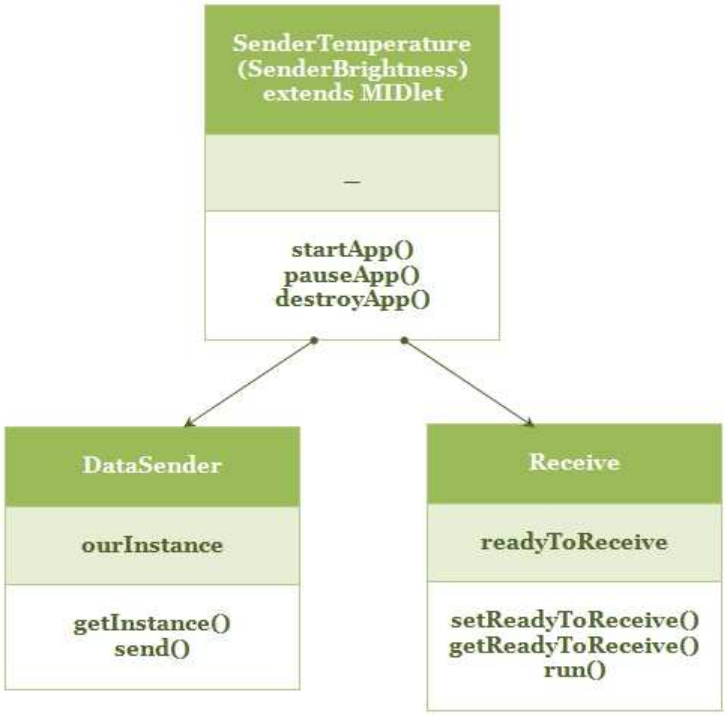
Οι δύο εφαρμογές **SenderTemperature** και **SenderBrightness**, δεν διαφέρουν καθόλου όσον αφορά τον κώδικα, παρά μόνο στο γεγονός ότι διαβάζουν από διαφορετικά αρχεία. Δηλαδή, παρουσιάζουν μόνο μία διαφορά σε μία γραμμή κώδικα, που έχει να κάνει με το άνοιγμα του αρχείου. Για το λόγο αυτό, θα περιγράψουμε μόνο την εφαρμογή **SenderTemperature**.

► **Εφαρμογή SenderTemperature:** Όπως αναφεράμε και προηγουμένως, και τα τρία SunSPOTs έχουν τη δυνατότητα να στείλουν και να λάβουν μηνύματα. Για το συγχρονισμό των δύο αυτών λειτουργιών έχουν χρησιμοποιηθεί διαφορετικά νήματα (threads), ώστε να υπάρχει συγχρονισμός μεταξύ αποστολής και λήψης. Η κύρια κλάση της εφαρμογής **SenderTemperature** περιλαμβάνει τις ακόλουθες κλάσεις:

- **DataSender**: η οποία χρησιμοποιείται για την αποστολή μηνυμάτων σε ένα άλλο SunSPOT. Η κλάση αυτή περιλαμβάνει μία μεταβλητή και δύο μεθόδους που περιγράφονται παρακάτω.
  - Η μεταβλητή **private static DataSender ourInstance**, η οποία όπως φαίνεται από τον ορισμό της έχει τον τύπο της κλάσης DataSender και χρησιμοποιείται από την μέθοδο getInstance(), που περιγράφεται αμέσως μετά. Η μεταβλητή αυτή είναι ουσιαστικά ένας δείκτης, που δείχνει σε ένα αντικείμενο DataSender.
  - Η μέθοδος **public static DataSender getInstance()**, η οποία όταν καλεστεί δημιουργείται ένα καινούριο αντικείμενο DataSender και επιστρέφεται η τιμή της μεταβλητής ourInstance (διεύθυνση μνήμης που αντιστοιχεί στο αντικείμενο που δείχνει). Αυτό βοηθάει στον συγχρονισμό για την αποστολή των μηνυμάτων σε ένα SunSPOT που «ακούει».
  - Η μέθοδος **synchronized public void send()** πραγματοποιεί την αποστολή μηνυμάτων μέσω διαγραμμάτων (datagrams) σε όποιο SunSPOT «ακούει».
- **Receive**: Η κλάση αυτή επεκτείνει την κλάση Thread και χρησιμοποιείται για τη λήψη μηνυμάτων από άλλα SunSPOTs. Στη συνέχεια, δίνεται μια σύντομη περιγραφή των μεταβλητών και των μεθόδων που περιλαμβάνει.
  - Η μεταβλητή **private int readyToReceive** έχει οριστεί για να ελέγχει αν το SunSPOT που πρόκειται να λάβει μήνυμα είναι έτοιμο για τη λήψη αυτή.

- Η μέθοδος **public void setReadyToReceive()** θέτει την μεταβλητή readyToReceive ίση με τη μονάδα, ώστε να ξεκινήσει η λήψη των μηνυμάτων.
- Η μέθοδος **public int getReadyToReceive()**: χρησιμοποιείται για το διάβασμα της τιμής που έχει η κρυφή μεταβλητή readyToReceive.
- Η μέθοδος **public void run()**, η οποία κληρονομείται από την κλάση Thread, χρησιμοποιείται για τη συγχρονισμένη λήψη των μηνυμάτων όταν το συγκεκριμένο SunSPOT, που τρέχει την εφαρμογή αυτή, ακούσει κάποιο άλλο SunSPOT να του στέλνει.

Τέλος, η κύρια κλάση **SenderTemperature** περιλαμβάνει και τις τρεις απαραίτητες μεθόδους για τη σωστή λειτουργία του MIDlet, δηλαδή, 1) τη μέθοδο **startApp**(), 2) τη μέθοδο **pauseApp**() και 3) τη μέθοδο **destroyApp**(). Παρακάτω δίνεται ένα σχηματικό διάγραμμα που απεικονίζει τις κλάσεις που περιγράφηκαν και στη συνέχεια δίνεται ο κώδικας που αναπτύχθηκε για την εφαρμογή **SenderTemperature**.



Σχήμα 26: *Κλάσεις της εφαρμογής SenderTemperature και SenderBrightness*

```
package net.java.dev.netbeansspot;

// Classes needed for the MIDlet
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.io.*;

// Class for Exception messages
import java.io.IOException;

// Classes for Reading a File
import java.io.InputStream;
import java.io.InputStreamReader;
import com.sun.squawk.io.BufferedReader;

// Class for using the method utils.sleep(t) to make a
// thread sleep for t ms ignoring any InterruptedException
import com.sun.spot.util.Utils;

// Class for sending a data message to another SunSPOT
class DataSender {
  // static instance(ourInstance) initialized as null.
  private static DataSender ourInstance = null;

  // Private constructor suppresses generation
  // of a default constructor.
  private DataSender() {
    // Does nothing
  }

  // DataSender is loaded on the first execution of
  // DataSender.getInstance() or the first access to
  // DataSender.ourInstance, not before. returns -> ourInstance
  public static DataSender getInstance() {
    synchronized (DataSender.class) {
      if (ourInstance == null) {
        ourInstance = new DataSender();
      }
    }
    return ourInstance;
  }

  // Send a message (String) to a specific Destination.
  synchronized public void send(final String targetAddress, final String msg) {
    try {
      // We create a DatagramConnection
      final DatagramConnection dgConnection = (DatagramConnection)
              Connector.open("radiogram://" + targetAddress + ":37");

      // Then, we ask for a datagram with the maximum size allowed
      final Datagram dg = dgConnection.newDatagram(dgConnection.getMaximumLength());

      //Ensures that the next read/write operation
      // will read/write from the start of the datagram
      dg.reset();
```

```
        //Write Data to Datagram
        dg.writeUTF(msg);

        //Send Datagram
        dgConnection.send(dg);

        //Close the connection
        dgConnection.close();

        Utils.sleep(500); // Make the thread sleep for 500ms
      } catch (IOException ex) {
        System.out.println("Could not open radiogram connection");
        ex.printStackTrace();
      } // end of try
    } // end of send method
} // end of class DataSender

// Class to receive messages
class Receive extends Thread{
  private int readyToReceive = 0;

  public void setReadyToReceive() {
    readyToReceive = 1;
  }

  public int getReadyToReceive() {
    return readyToReceive;
  }

  public void run() {
    //Setting up the Datagram Connection
    DatagramConnection dgConnection = null;
    Datagram dg = null;

    try {
      //Open Datagram Connection on port 37
      dgConnection = (DatagramConnection) Connector.open("radiogram://:37");

      // Then, we ask for a datagram with the maximum size allowed
      dg = dgConnection.newDatagram(dgConnection.getMaximumLength());

    } catch (IOException e) {
      System.out.println("Could not open radiogram receiver connection");
      e.printStackTrace();
      return;
    } // end of try

    while (true) {
      try {
        // Ensures that the next read or write operation
        // will read/write from the start of the radiogram
        dg.reset();

        //Receive a Datagram
        dgConnection.receive(dg);
```

```
          //Read a String from datagram.
          final String rcvMsg = dg.readUTF();

          if (rcvMsg.equalsIgnoreCase("ok")) {
            setReadyToReceive();
          } else {
            System.out.println("Not ready to receive yet!");
          }

        } catch (IOException e) {
          System.out.println("Nothing received");
          e.printStackTrace();
        } // end of try
      } // end of while(true)
    } // end of method run()
  } // end of class Receive

  public class SenderTemperature extends MIDlet {

    protected void startApp() throws MIDletStateChangeException {
      // Read from file temperature.txt
      InputStream inputStream = getClass().getResourceAsStream("/temperature.txt");
      InputStreamReader dataReader = new InputStreamReader(inputStream);
      BufferedReader bufferedReader = new BufferedReader(dataReader);
      String thisLine = null;
      StringBuffer data = new StringBuffer();
      final String destination = "7f00.0001.0000.1003";
      Receive spot = new Receive();

      // Flush the input stream from leftover characters
      System.out.flush();

      spot.start(); //Start to Receive

      while (true) {
        if (spot.getReadyToReceive() == 1) {
          try {
            while ((thisLine = bufferedReader.readLine()) != null) {
              data.append(thisLine + "\n");
              DataSender.getInstance().send(destination, thisLine);
            }
          } catch(Exception e) {
            e.printStackTrace();
          } // end of try
        } // end of if statement
      } // end of while(true)
    }

    protected void pauseApp() {
      // This is not currently called by the Squawk VM
    }

    protected void destroyApp(boolean unconditional) throws
        MIDletStateChangeException { // TODO
    }
  } // end of class SenderTemperature
```

► **Εφαρμογή CrosierOnlineAppReceiver:** Η κύρια κλάση της εφαρμογής αυτής περιλαμβάνει τις ίδιες κλάσεις με αυτές που περιγράφηκαν για την εφαρμογή **CrosierApplication** στην παράγραφο 6.3.2 και έχουν προστεθεί και οι κλάσεις DataSender και Receive για την αποστολή και λήψη μηνυμάτων, οι οποίες περιγράφηκαν στην παράγραφο 6.3.3. Επίσης, στην κλάση CrosierAlgorithm που περιγράφεται ο αλγόριθμος Crosier, έχουν προστεθεί κάποιες μέθοδοι. Παρακάτω δίνεται μια σύντομη λίστα και περιγραφή των στοιχείων που περιλαμβάνει η κύρια κλάση της εφαρμογής **CrosierOnlineAppReceiver**.

- **CrosierState**: η κλάση αυτή έχει περιγραφεί στην παράγραφο 6.3.2.
- **CrosierAlgorithm**: η κλάση αυτή περιλαμβάνει τις ίδιες μεταβλητές και μεθόδους όπως έχουν περιγραφεί στην παράγραφο 6.3.2 και έχουν προστεθεί κάποιες μεταβλητές και μέθοδοι, που περιγράφονται στη συνέχεια. Τις μεταβλητές και τις μεθόδους αυτές τις προσθέσαμε για δύο λόγους. Πρώτον, για την αρχικοποίηση του συστήματος και δεύτερον για την ανανέωση των στατιστικών στοιχείων που υπολογίζονται. Η αρχικοποίηση του συστήματος γίνεται μετά από τη λήψη 50 δειγμάτων για τη θερμοκρασία και τη φωτεινότητα, όπου υπολογίζονται όλα τα στατιστικά στοιχεία που χρειάζονται για τον αλγόριθμο Crosier. Έπειτα, με τις τιμές αυτές η εφαρμογή ελέγχει την κατάσταση του συστήματος μέσω του αλγορίθμου Crosier. Στη συνέχεια, ο αλγόριθμος Crosier τρέχει με τις τιμές που υπολογίστηκαν στην αρχικοποίηση, ενώ παράλληλα ανανεώνονται τα στατιστικά στοιχεία για κάθε καινούριο ζεύγος τιμών (θερμοκρασία, φωτεινότητα), χωρίς το δεύτερο να επηρεάζει το πρώτο. Τα στατιστικά στοιχεία ανανεώνονται κάθε φορά που ολοκληρώνεται ένα σετ από 50 δείγματα ζεύγων (θερμοκρασίας, φωτεινότητας).
  - Η μέθοδος **public void meanUpdate()**: έχει οριστεί ώστε να κάνει μια απλή αντικατάσταση των καινούριων μέσων τιμών και να ανανεώνει το διάνυσμα των μέσων **newMeans**, αφού ολοκληρωθούν όλοι οι υπόλοιποι στατισ-

τικοί υπολογισμοί. Ο λόγος που γίνεται αυτό είναι γιατί όπως θα δούμε και στις επόμενες σχέσεις, για την online ανανέωση των σταστικών στοιχείων χρειάζονται και οι παλιές αλλά και οι καινούριες μέσες τιμές.

– Η μέθοδος **public void findNewMeans()**: χρησιμοποιείται για τον υπολογισμό των καινούριων μέσων τιμών, σύμφωνα με τη σχέση,

$$\bar{x}_{new} = \bar{x}_{old} + \frac{x_n - \bar{x}_{old}}{n}$$

όπου x_n είναι η καινούρια τιμή της μεταβλητής x που λαμβάνει ο αλγόριθμος.

– Η μέθοδος **public void updateVariances()**: ανανεώνει τις τιμές των διακυμάνσεων των δύο μεταβλητών (θερμοκρασία, φωτεινότητα), σύμφωνα με τη σχέση,

$$var(x)_{new} = \frac{var(x)_{old} + (x_n - \bar{x}_{old})(x_n - \bar{x}_{new})}{n-1}$$

όπου x_n είναι η καινούρια τιμή της μεταβλητής x που λαμβάνει ο αλγόριθμος.

– Η μέθοδος **public void updateCovariance()**: υπολογίζει την καινούρια συνδιακύμανση των δύο μεταβλητών (θερμοκρασία, φωτεινότητα), σύμφωνα με τη σχέση,

$$cov(x,y)_{new} = \frac{(n-1)cov(x,y)_{old} + \frac{n-1}{n}(x_n - \bar{x}_{old})(y_n - \bar{y}_{old})}{n}$$

όπου x_n και y_n είναι οι καινούριες τιμές των μεταβλητών x και y που λαμβάνει ο αλγόριθμος.

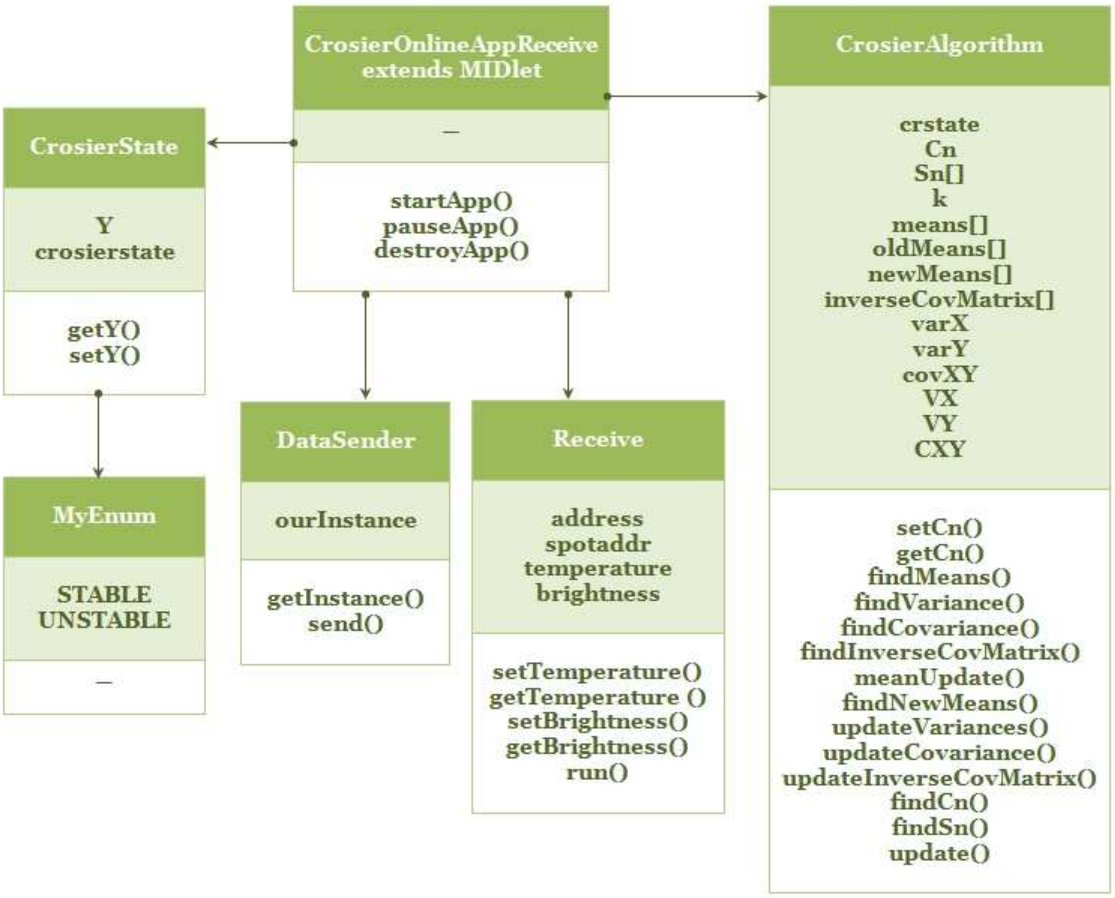
– Η μέθοδος **public void updateInverseCovMatrix()**: ανανεώνει τον πίνακα συνδιακύμανσης με τις καινούριες τιμές που υπολογίστηκαν.

- **DataSender**: η κλάση αυτή έχει περιγραφεί στην παράγραφο 6.3.3.
- **Receive**: η κλάση αυτή είναι λίγο διαφορετική από την κλάση που περιγράφηκε στην παράγραφο 6.3.3. Αυτό οφείλεται στο γεγονός ότι στην κλάση αυτή θα

τρέξει ο αλγόριθμος με τις τιμές που θα λάβει από τα δύο SunSPOTs. Έτσι, έχουν οριστεί κάποιες επιπλέον μεταβλητές και η μέθοδος run() που κληρονομείται από την κλάση Thread έχει διαμορφωθεί κατάλληλα για την εφαρμογή αυτή και περιγράφεται αναλυτικά στη συνέχεια.

- Η μεταβλητή **String address**: έχει οριστεί για να αποθηκεύεται η διεύθυνση του SunSPOT, από το οποίο γίνεται η λήψη της επόμενης μέτρησης.
- Η μεταβλητή **char spotaddr**: είναι ίση με τον τελευταίο χαρακτήρα της μεταβλητής address, και χρησιμοποιείται σε μία συνθήκη switch, ώστε να γίνεται η διάκριση ανάμεσα στα δύο SunSPOTs. Με αυτόν τον τρόπο η εφαρμογή μας καταλαβαίνει αν η μέτρηση που μόλις έχει λάβει είναι τιμή θερμοκρασίας ή τιμή φωτεινότητας.
- Η μεταβλητή **private double temperature**: χρησιμοποιείται για να αποθηκεύσει την καινούρια τιμή θερμοκρασίας που λαμβάνει η εφαρμογή.
- Η μεταβλητή **private double brightness**: χρησιμοποιείται για να αποθηκεύσει την καινούρια τιμή φωτεινότητας που λαμβάνει η εφαρμογή.
- Η μέθοδος **synchronized public void run()**: ξεκινάει τη σύνδεση με όποιο SunSPOT «ακούσει», κάνει τον έλεγχο με τη συνθήκη switch, όπως περιγράψαμε παραπάνω, για να καταλάβει αν λαμβάνει τιμή θερμοκρασίας ή τιμή φωτεινότητας και τρέχει έπειτα τον αλγόριθμο Crosier, αφού λάβει τιμές και για τις δύο μεταβλητές (θερμοκρασία, φωτεινότητα).

Τέλος, η κύρια κλάση **CrosierOnlineAppReceiver** περιλαμβάνει και τις τρεις απαραίτητες μεθόδους για τη σωστή λειτουργία του MIDlet, δηλαδή, 1) τη μέθοδο **startApp**(), 2) τη μέθοδο **pauseApp**() και 3) τη μέθοδο **destroyApp**(). Παρακάτω δίνεται ένα σχηματικό διάγραμμα που απεικονίζει τις κλάσεις που περιγράφηκαν και στη συνέχεια δίνεται ο κώδικας που αναπτύχθηκε για την εφαρμογή **CrosierOnlineAppReceiver**.



Σχήμα 27: *Κλάσεις της εφαρμογής CrosierOnlineAppReceiver*

```
package net.java.dev.netbeansspot;

// Classes needed for the MIDlet
import javax.microedition.io.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

// Class for Exception messages
import java.io.IOException;

// Class to define the system's state
class CrosierState {
  private double Y; // value that defines the alarm
  String crosierstate; // to remember previous state

  /* TWO states: STABLE, UNSTABLE                    */
  /* STABLE    ---> Temperature under the limit      */
  /* UNSTABLE  ---> Temperature over the limit       */
  public class MyEnum {
    public final static String STABLE = "STABLE";
    public final static String UNSTABLE = "UNSTABLE";
  }

  // Default constructor
  public CrosierState() {
    this.Y = 0.0;
    crosierstate = "STABLE";
  }

  // Method for getting the value of Y
  public double getY() {
    return Y;
  }

  // Method for setting a new value to Y
  public void setY(double Y) {
    this.Y = Y;
  }
}

// Class for the CrosierAlgorithm where the output depends on
//  the two dimension vector x = (temperature, brightness)
class CrosierAlgorithm {
  // Variables for the Crosier Algorithm
  private CrosierState crstate = null;
  private double Cn;
  double[] Sn = new double[2];
  double k;

  // Variables for the statistical formulas
  int n = 0; // number of sample values
  double[] means = new double[2]; // holds the first mean values
  double[] inverseCovMatrix = new double[4]; // the inverse covariance matrix
  double varX; // var(X)
  double varY; // var(Y)
  double covXY; // cov(X,Y)

  // Temporary variables for the updating of the statistical elements
  double[] oldMeans = new double[2]; // holds the old mean values
  double[] newMeans = new double[2]; // for online updating the means
```

```
double VX; // var(X)
double VY; // var(Y)
double CXY; // cov(X,Y)
double[] INCOV = new double[4]; // inverse covariance matrix

// Default constructor
public CrosierAlgorithm() {
  crstate = new CrosierState();
  this.Cn = 0.0;
}

// Constructor with  three parameters
public CrosierAlgorithm(double k, double[] meanMatrix, double[] sn) {
  this.means = meanMatrix;
  this.Sn = sn;
  this.k = k;
  crstate = new CrosierState();
}

// Constructor with one parameter
public CrosierAlgorithm(double k) {
  this.k = k;
  this.Cn = 0.0;
  crstate = new CrosierState();
}

// Setter and getter for the private variable Cn
public void setCn(double cn) {
  this.Cn = cn;
}
public double getCn() {
  return this.Cn;
}

/* The following methods are used to calculate and update */
/* all the statistical elemenents that are necessary for  */
/* the Crosier algorithm: 1)mean vector, 2)variance,      */
/* 3)covariance and 4)inverse covariance matrix           */

// Method for finding the mean vector means[]
public void findMeans(double[] x, double[] y) {
  double tempx = 0.0;
  double tempy = 0.0;

  n = x.length; // n = number of sample values

  for (int i=0; i < n; i++) {
    tempx = tempx + x[i];
    tempy = tempy + y[i];
  }
  tempx /= n;
  tempy /= n;

  means[0] = tempx; // mean(x)
  means[1] = tempy; // mean(y)
} // end of method findMeans()
```

```
// Method for finding the variance of a variable x
public double findVariance (double[] x, double meanX) {
  double[] temp = new double[n]; // n values of x
  double variance = 0.0;

  for (int i = 0; i < n; i ++) {
    temp[i] = x[i] - meanX;
    variance += temp[i] * temp[i];
  }
  variance /= (n-1);
  return variance;
} // end of method findVariance

// Method for finding the covariance between two variables x and y
public double findCovariance (double[] x, double[] y, double meanX, double meanY) {
  double[] tempX = new double[n];
  double[] tempY = new double[n];
  double covariance = 0.0;

  for (int i = 0; i < n; i ++) {
    tempX[i] = x[i] - meanX;
    tempY[i] = y[i] - meanY;
    covariance += tempX[i] * tempY[i];
  }
  covariance /= (n-1);
  return covariance;
} // end of method findCovariance

// Method for finding the inverse covariance matrix for two variables x and y
public void findInverseCovMatrix(double[] x, double[] y, double meanX, double meanY) {
  double[] covMatrix = new double[4];
  double[] tempCovMatrix = new double[4];
  double det = 0.0;

  varX = findVariance(x, meanX);
  varY = findVariance(y, meanY);
  covXY = findCovariance(x, y, meanX, meanY);

  covMatrix[0] = varX;
  covMatrix[1] = covXY;
  covMatrix[2] = covXY;
  covMatrix[3] = varY;

  // find the determinant
  det = covMatrix[0]*covMatrix[3] - covMatrix[1]*covMatrix[2];

  if (det != 0) {
    tempCovMatrix[0] = covMatrix[3]/det;
    tempCovMatrix[1] = -covMatrix[1]/det;
    tempCovMatrix[2] = -covMatrix[2]/det;
    tempCovMatrix[3] = covMatrix[0]/det;
    inverseCovMatrix = tempCovMatrix;
  } else {
    System.out.println("The determinant is zero!!!");
  }
} // end of method findInverseCovMatrix

// Method for updating the means
public void meanUpdate() {
  oldMeans[0] = newMeans[0];
  oldMeans[1] = newMeans[1];
}
```

```
// Method for calculating the new mean of a variable
public void findNewMeans(double[] xn) {
  n = n + 1;
  newMeans[0] = oldMeans[0] + (xn[0] - oldMeans[0])/n;
  newMeans[1] = oldMeans[1] + (xn[1] - oldMeans[1])/n;
}

// Method for updating the new variances of x and y
public void updateVariances(double x, double y) {
  double tempX = 0.0;
  double tempY = 0.0;

  tempX = VX + (x - oldMeans[0])*(x - newMeans[0]);
  tempY = VY + (y - oldMeans[1])*(y - newMeans[1]);

  VX = tempX/(n-1);
  VY = tempY/(n-1);
} // end of method updateVariances

// Method for updating the covariance of x and y
public void updateCovariance(double x, double y) {
  double tempCov = 0.0;
  double tempXY = 0.0;

  tempXY = (n-1)*(x - oldMeans[0])*(y - oldMeans[1])/n;
  tempCov = CXY *(n-1) + tempXY;
  CXY = tempCov/n;
} // end of method updateCovariance

// Method for updating the inverse covariance matrix
public void updateInverseCovMatrix() {
  double[] covMatrix = new double[4];
  double[] tempCovMatrix = new double[4];
  double det = 0.0;

  covMatrix[0] = VX;
  covMatrix[1] = CXY;
  covMatrix[2] = CXY;
  covMatrix[3] = VY;

  det = covMatrix[0]*covMatrix[3] - covMatrix[1]*covMatrix[2];

  if (det != 0) {
    tempCovMatrix[0] = covMatrix[3]/det;
    tempCovMatrix[1] = -covMatrix[1]/det;
    tempCovMatrix[2] = -covMatrix[2]/det;
    tempCovMatrix[3] = covMatrix[0]/det;
    INCOV = tempCovMatrix;
  } else {
    System.out.println("The determinant is zero!!!");
  }
} // end of updateInverseCovMatrix

/* The following methods are used to calculate the final result Y, */
/* according to the bivariate cusum algorithm described by Crosier. */
```

```
// Method for finding the new value of Cn
public void findCn(double[] xn, double[] covMat) {
  double c = 0.0;
  double[] temp_cn = new double[2];
  double[] temp_mult = new double[2];

  temp_cn[0] = Sn[0] + xn[0] - means[0];
  temp_cn[1] = Sn[1] + xn[1] - means[1];
  temp_mult[0] = temp_cn[0] * covMat[0] + temp_cn[1] * covMat[2];
  temp_mult[1] = temp_cn[0] * covMat[1] + temp_cn[1] * covMat[3];
  c = temp_mult[0] * temp_cn[0] + temp_mult[1] * temp_cn[1];

  if (c >= 0) {
      setCn(Math.sqrt(c));
  }
  System.out.println("Cn = " + getCn());

  if (getCn() <= k) {
      Sn[0] = 0;
      Sn[1] = 0;

      for(int i=0; i<2; i++) {
          System.out.println("Sn["+i+"] = "+Sn[i]);
      }
  } else {
      findSn(getCn(), xn);
  }
} // end of method findCn

// Method for finding the new vector Sn
public void findSn(double cn, double[] xn) {
  double c = 0.0;
  double[] temp_sn = new double[2];

  c = 1 - k/getCn();
  temp_sn[0] = c * (Sn[0] + xn[0] - means[0]);
  temp_sn[1] = c * (Sn[1] + xn[1] - means[1]);
  Sn = temp_sn;

  for(int i=0; i<2; i++) {
    System.out.println("Sn["+i+"] = "+Sn[i]);
  }
} // end of findSn

// Method that defines the final output of the algorithm
public int update(double[] covMat, double threshold) {
  int output = 0;
  double yn = 0.0;
  double[] temp_yn = new double[2];

  temp_yn[0] = Sn[0] * covMat[0] + Sn[1] * covMat[2];
  temp_yn[1] = Sn[0] * covMat[1] + Sn[1] * covMat[3];

  yn = Math.sqrt(temp_yn[0] * Sn[0] + temp_yn[1] * Sn[1]);
  System.out.println("Yn = "+yn);

  if (yn < threshold) {
    if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.STABLE)) {
      output = 0;
    }
    else if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.UNSTABLE)) {
      crstate.crosierstate = CrosierState.MyEnum.STABLE;
      output = -1;
    }
```

```
      } else {
        if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.STABLE)) {
          crstate.crosierstate = CrosierState.MyEnum.UNSTABLE;
          output = 1;
        }
        else if (crstate.crosierstate.equalsIgnoreCase(CrosierState.MyEnum.UNSTABLE)) {
          output = 0;
        }
      }
      return output;
    } // end of method update()
  } // end of class CrosierAlgorithm

  // DataSender is loaded on the first execution of
  // DataSender.getInstance() or the first access to
  // DataSender.ourInstance, not before. Returns -> ourInstance
  class DataSender {

    // static instance(ourInstance) initialized as null.
    private static DataSender ourInstance = null;

    // Private constructor suppresses generation of a default constructor.
    private DataSender() {
      // Does nothing
    }

    public static DataSender getInstance() {
      synchronized (DataSender.class) {
        if (ourInstance == null) {
          ourInstance = new DataSender();
        }
      }
      return ourInstance;
    } // end of method getInstance()

    // Send a message (String) to a specific Destination.
    public void send(final String targetAddress, final String msg) {
      try {
        // We create a DatagramConnection
        final DatagramConnection dgConnection = (DatagramConnection)
                Connector.open("radiogram://" + targetAddress + ":37");

        // Then, we ask for a datagram with the maximum size allowed
        final Datagram dg = dgConnection.newDatagram(dgConnection.getMaximumLength());

        // Ensures that the next read/write operation will
        // read/write from the start of the datagram
        dg.reset();

        //Write Data to Datagram
        dg.writeUTF(msg);

        //Send Datagram
        dgConnection.send(dg);

        //Close the connection
        dgConnection.close();

      } catch (IOException ex) {
        System.out.println("Could not open radiogram connection");
        ex.printStackTrace();
      } // end of try
    } // end of method send()
  } // end of class DataSender
```

```
// Class to receive messages
class Receive extends Thread {
  String address;
  char spotaddr;
  private double temperature;
  private double brightness;

  // Getters and Setters for private variables
  // temperature and brightness
  public void setTemperature(double temp) {
    this.temperature = temp;
  }
  public double getTemperature() {
    return temperature;
  }
  public void setBrightness(double br) {
    this.brightness = br;
  }
  public double getBrightness() {
    return brightness;
  }

  // Method run is synchronized so that any changes that happens
  // to the state of the algorithm will be visible to all threads.
  synchronized public void run() {

    //Setting up the Datagram Connection
    DatagramConnection dgConnection = null;
    Datagram dg = null;

    // Make a new CrosierAlgorithm object with k = 0.5
    CrosierAlgorithm crosier = new CrosierAlgorithm(0.5);

    // variables that will be useful to loop statements
    int count2 = 0;
    int count = 1;
    int i = 0;

    // Variables that will be used for the algorithm
    int outValue = 0;
    double newTemperature = 0.0;
    double newBrightness = 0.0;
    double[] tempVector = new double[51];    // temperature buffer
    double[] brightVector = new double[51]; // brightness buffer
    double[] vector = new double[2]; // vector = (temperature,brightness)
    double[] s0 = {0.0, 0.0}; // for the initialization of vector Sn
    double h = 5.5; // threshold

    try {
      //Open Datagram Connection on port 37
      dgConnection = (DatagramConnection) Connector.open("radiogram://:37");

      // Then, we ask for a datagram with the maximum size allowed
      dg = dgConnection.newDatagram(dgConnection.getMaximumLength());

    } catch (IOException e) {
      System.out.println("Could not open radiogram receiver connection");
      e.printStackTrace();
      return;
    } // end of try
```

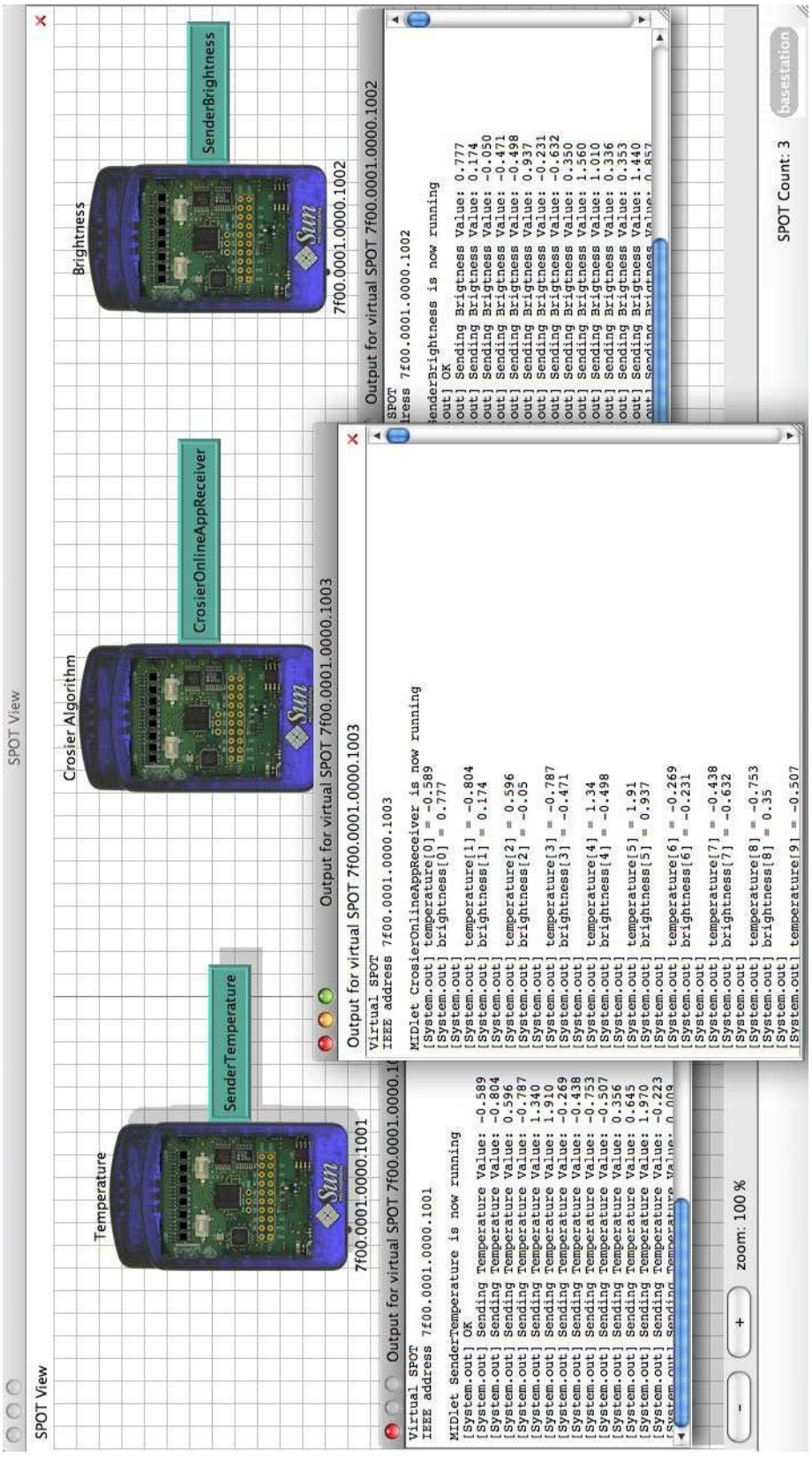
```
while (true) {
  try {
    // Ensures that the next read or write operation
    // will read/write from the start of the radiogram
    dg.reset();

    //Receive a Datagram
    dgConnection.receive(dg);

    // Get the address of the sender
    address = dg.getAddress();

    // Check the last character of the address to
    // see which of the two spots is sending the dg
    spotaddr = address.charAt(18);

    //Read a String from datagram.
    final String rcvMsg = dg.readUTF();
    double temp = Double.parseDouble(rcvMsg);

    // Depending on the spot's address the SunSPOT that runs
    // this application will know if the data that is receiving
    // represent values of temperature or brightness
    switch(spotaddr) {

      case '1':  // case the sender's address last digit is 1
         setTemperature(temp);
         break;

      case '2':  // case the sender's address last digit is 2
         setBrightness(temp);
         break;

      default: System.out.println("Something went terribly wrong!");
    } // end of switch
  } catch (IOException e) {
    System.out.println("Nothing received");
    e.printStackTrace();
  } // end of try
  ++count2;

  // The next if statement will be entered whenever the SunSPOT
  // running this application receives new value samples for both
  // the temperature and brightness.
  if (count2%2 == 0) {

    // Initialization for vector Sn
    if (count == 1) {
      crosier.Sn = s0;
    }

    // The next if statement collects the first 50 value samples
    // for temperature and brightness to calculate all the
    // statistical elements and run the Crosier algorithm.
    if (count <= 50) {
      // Save the samples in buffers
      tempVector[i] = getTemperature();
      brightVector[i] = getBrightness();
      System.out.println("temperature["+i+"] = " + tempVector[i]);
      System.out.println("brightness["+i+"] = " + brightVector[i]);
      System.out.println();
```

```
    // Calculations for the statistical elements
    // after collecting the first  value samples
    if (count == 50) {
      // Get the new temperature and brightness value samples
      newTemperature = getTemperature();
      newBrightness = getBrightness();

      // Place the new values into vector mode
      vector[0] = newTemperature;
      vector[1] = newBrightness;

      // Calculate the means for temperature and brightness
      crosier.findMeans(tempVector, brightVector);

      // Find the inverse covariance matrix
      crosier.findInverseCovMatrix(tempVector,brightVector,
                    crosier.means[0],crosier.means[1]);

      // Initialize the temporary statistical variables
      crosier.oldMeans = crosier.means;
      crosier.VX = crosier.varX;
      crosier.VY = crosier.varY;
      crosier.CXY = crosier.covXY;

      // Print all the statistical elements
      System.out.println();
      System.out.println("---- System's Statistical Elements ----");
      System.out.println();
      System.out.println("X: Temperature, Y: Brightness");
      System.out.println("mean(X) = " + crosier.means[0]);
      System.out.println("mean(Y) = " + crosier.means[1]);
      System.out.println("var(X) = " + crosier.varX);
      System.out.println("var(Y) = " + crosier.varY);
      System.out.println("cov(X,Y) = " + crosier.covXY);

      for (int z = 0; z < 4; z++) {
          System.out.println("invCovMatrix["+z+"] = "
                    + crosier.inverseCovMatrix[z]);
      }
      System.out.println();
      System.out.println("---- System's Statistical Elements ----");
      System.out.println();

      // Print the results
      System.out.println("------ Results for samples n = "+count+" ------");
      System.out.println("(temperature, brightness) = ("
              +newTemperature+", "+newBrightness+")");
      System.out.println();
      // Run the Crosier Algorithm with threshold h = 5.5
      crosier.findCn(vector, crosier.inverseCovMatrix);
      outValue = crosier.update(crosier.inverseCovMatrix, h);
      System.out.println();
      System.out.println("OUTPUT = " + outValue);
      System.out.println("-------------------------------------------------------");
      System.out.println();
    } // end if (count == 50)
    i++;
  } // end if (count <= 50)
```

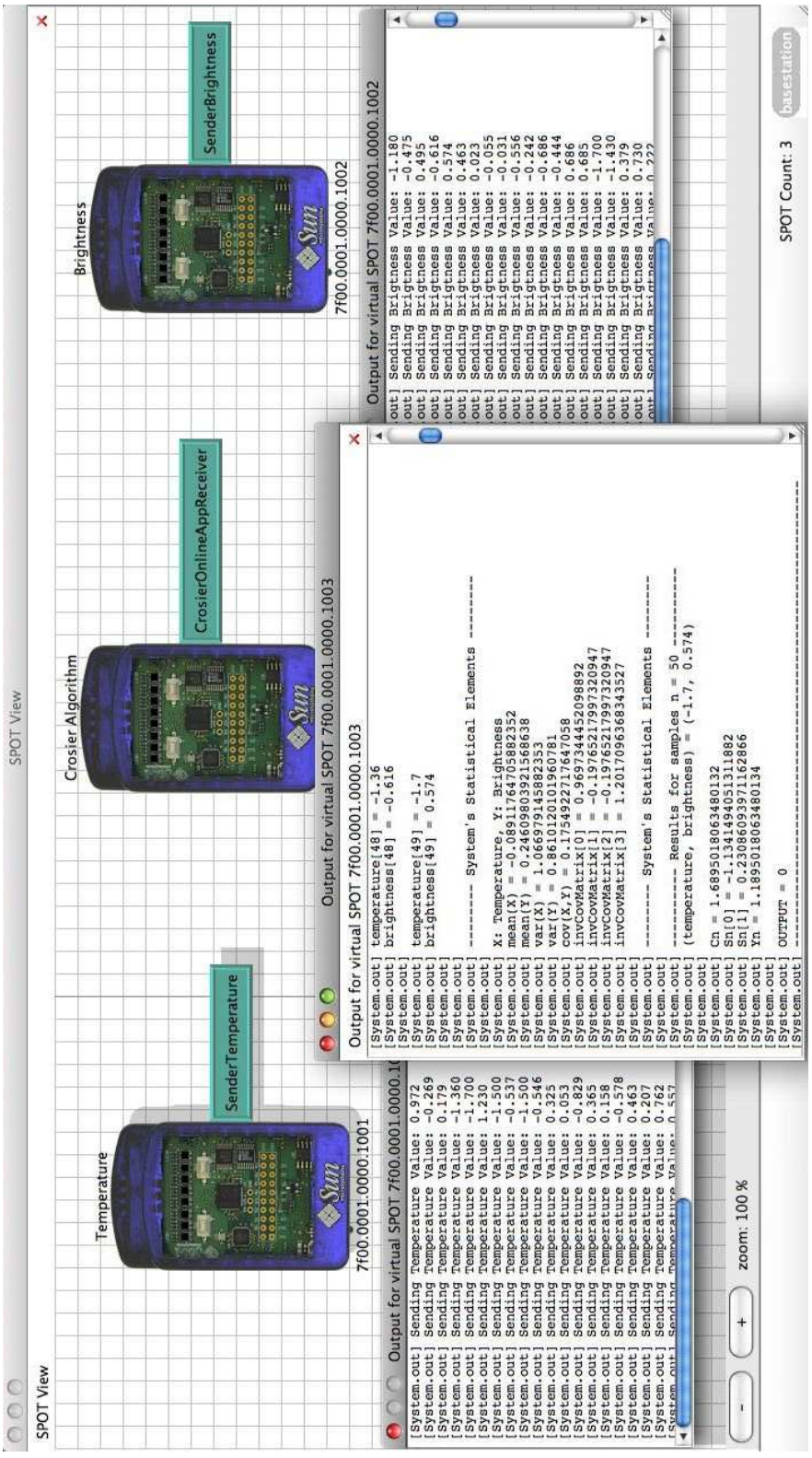
```
else if (count > 50) {
    // Get the new temperature and brightness value samples
    newTemperature = getTemperature();
    newBrightness = getBrightness();
    vector[0] = newTemperature;
    vector[1] = newBrightness;

    // We use the temporary variables of the crosier algorithm
    // for the online updating of all the statistical elements.
    // The Crosier algorithm, though, runs with the old statistical
    // values (mean, covariance matrix), and after 50 new samples
    // the online updating refreshes the values and the algorithm
    // runs with the new means and new covariance matrix. That happens
    // every 50 new sample values.
    crosier.findNewMeans(vector);
    crosier.updateVariances(vector[0], vector[1]);
    crosier.updateCovariance(vector[0], vector[1]);
    crosier.updateInverseCovMatrix();
    crosier.meanUpdate();

    // After 50 value samples -> Update all the statistical elements
    if (count % 50 == 0) {
      crosier.means = crosier.newMeans;
      crosier.varX = crosier.VX;
      crosier.varY = crosier.VY;
      crosier.covXY = crosier.CXY;
      crosier.inverseCovMatrix = crosier.INCOV;

      // Print all the updated statistical elements
      System.out.println();
      System.out.println("---- System's Updated Statistical Elements ----");
      System.out.println();
      System.out.println("X: Temperature, Y: Brightness");
      System.out.println("mean(X) = " + crosier.means[0]);
      System.out.println("mean(Y) = " + crosier.means[1]);
      System.out.println("var(X) = " + crosier.varX);
      System.out.println("var(Y) = " + crosier.varY);
      System.out.println("cov(X,Y) = " + crosier.covXY);
      for (int z = 0; z < 4; z++) {
        System.out.println("invCovMatrix["+z+"] = "
                +crosier.inverseCovMatrix[z]);
      }
      System.out.println();
      System.out.println("-----------------------------------------------------");
```

```
            System.out.println();

            // Online update for the means
            crosier.meanUpdate();
          } // end if (count % 50 == 0)

          // Print the results
          System.out.println("----- Results for samples n = "+count+" -----");
          System.out.println("(temperature, brightness) = ("
                  +newTemperature+","+newBrightness+")");
          System.out.println();

          // Run the Crosier Algorithm with threshold h = 5.5
          crosier.findCn(vector, crosier.inverseCovMatrix);
          outValue = crosier.update(crosier.inverseCovMatrix, h);
          System.out.println();
          System.out.println("OUTPUT = " + outValue);
          System.out.println("-------------------------------------");
          System.out.println();
        }// end if (count > 50)
        count++;
      } // end if (count2%2 == 0)
    } // end while (true)
  } // end sunchronized public void run()
} // end class Receive

public class CrosierOnlineAppReceiver extends MIDlet {

  protected void startApp() throws MIDletStateChangeException {
    final String spot1 = "7f00.0001.0000.1001";
    final String spot2 = "7f00.0001.0000.1002";

    //Start receiving
    Receive receiver = new Receive();
    receiver.start();

    // Send "OK" to spot1 and spot2 so that they will
    // start sending the value samples
    try {
      DataSender.getInstance().send(spot1, "OK");
      DataSender.getInstance().send(spot2, "OK");
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  protected void pauseApp() {
    // This is not currently called by the Squawk VM
  }

  protected void destroyApp(boolean unconditional) throws
      MIDletStateChangeException { // TODO
  }
}
```

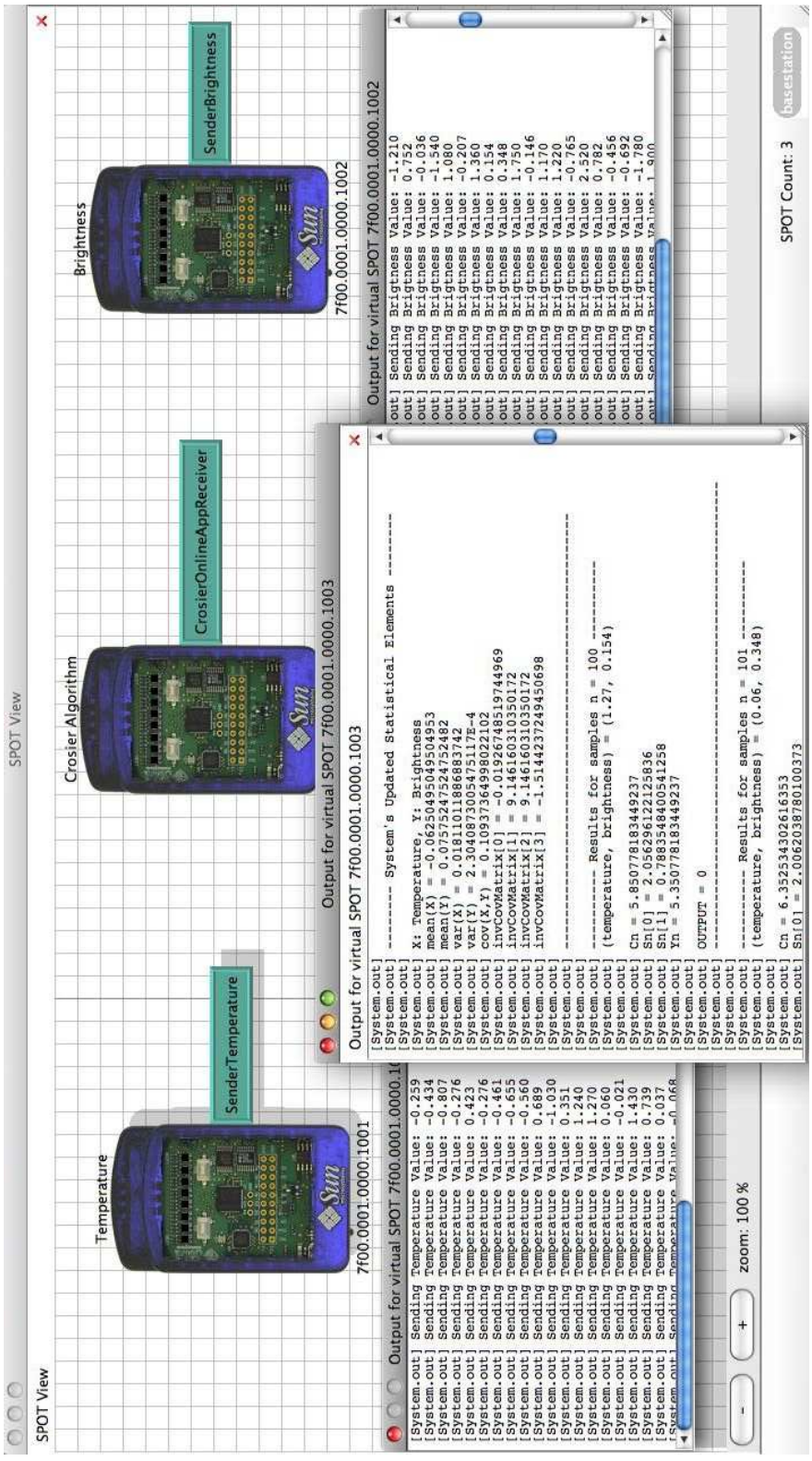
► **Αποτελέσματα Εφαρμογής CrosierOnlineAppReceiver:** Για τον έλεγχο της ορθότητας της εφαρμογής CrosierOnlineAppReceiver, επιλέξαμε 200 τυχαίες τιμές που ανήκουν στην κανονική κατανομή $N \sim (0, 1)$ διαφορετικές για τη θερμοκρασία και διαφορετικές για τη φωτεινότητα, τις οποίες αποθηκεύσαμε στα αντίστοιχα αρχεία temperature.txt και brightness.txt. Το κάθε αρχείο είναι αποθηκευμένο στον αντίστοιχο φάκελο resources του SunSPOT που πρόκειται να το διαβάσει. Στη συνέχεια, δίνονται τα αποτελέσματα που πήραμε από το τρέξιμο της εφαρμογής CrosierOnlineAppReceiver πάνω στο δίκτυο των SunSPOTs.



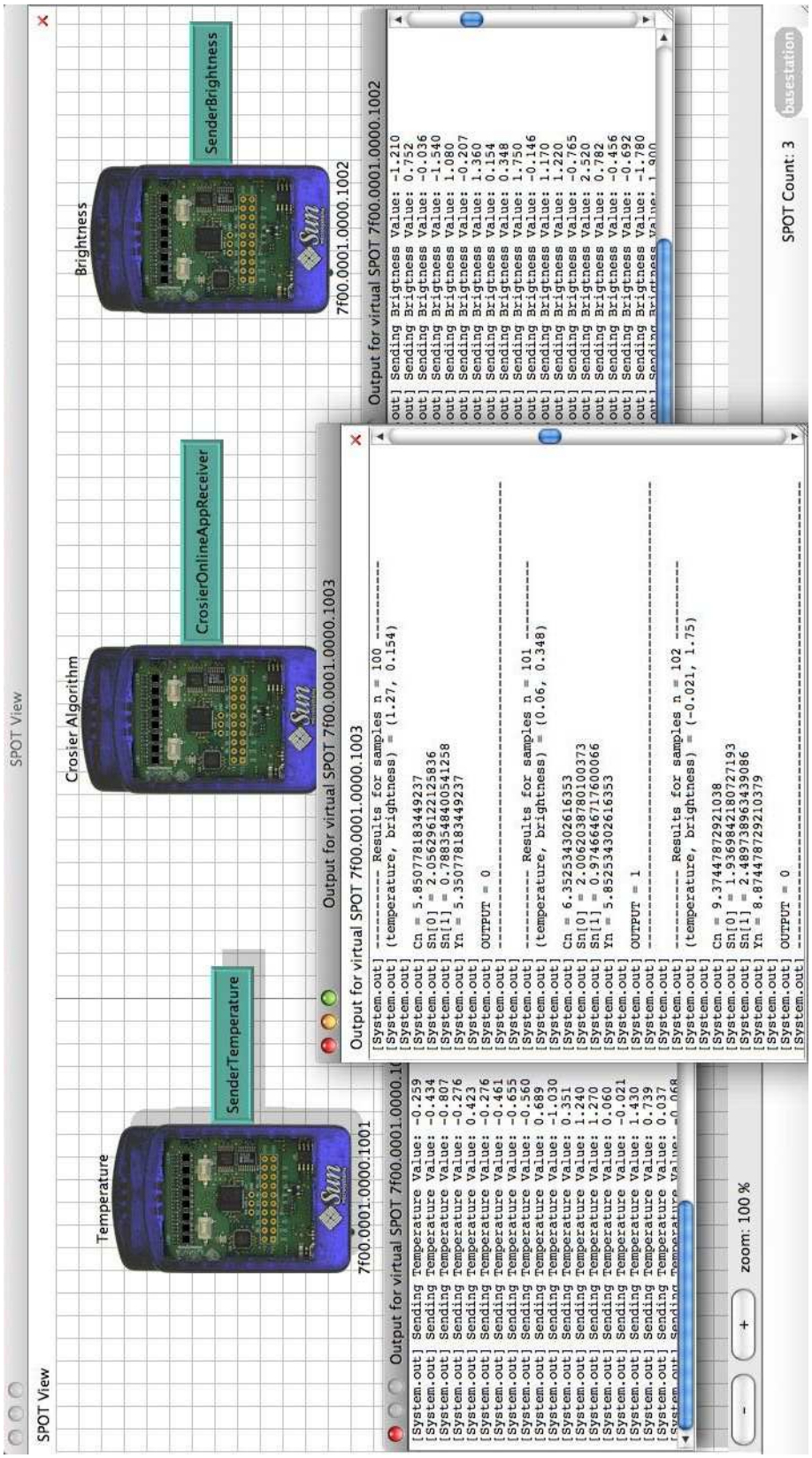
Σχήμα 28: *Λήψη των πρώτων 50 δειγμάτων (θερμοκρασίας, φωτεινότητας από τα SunSPOTs*



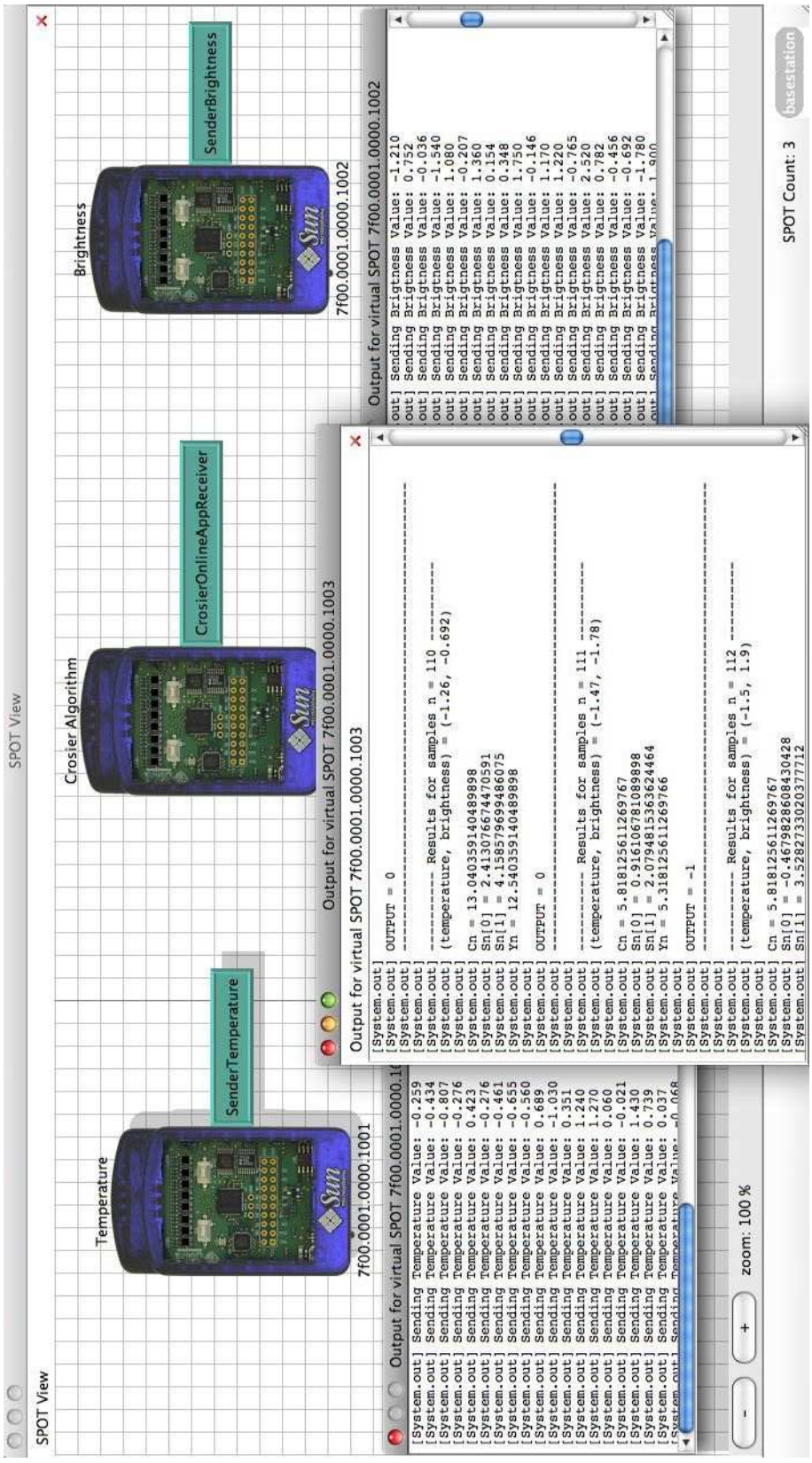
Σχήμα 29: *Ολοκλήρωση της λήψης όλων των 50 δειγμάτων (θερμοκρασίας, φωτεινότητας) από τα SunSPOTs, υπολογισμός όλων των στατιστικών στοιχείων (μέσες τιμές, διακυμάνσεις, πίνακας συνδιακύμανσης) και πρώτα αποτελέσματα αλγορίθμου Crosier*



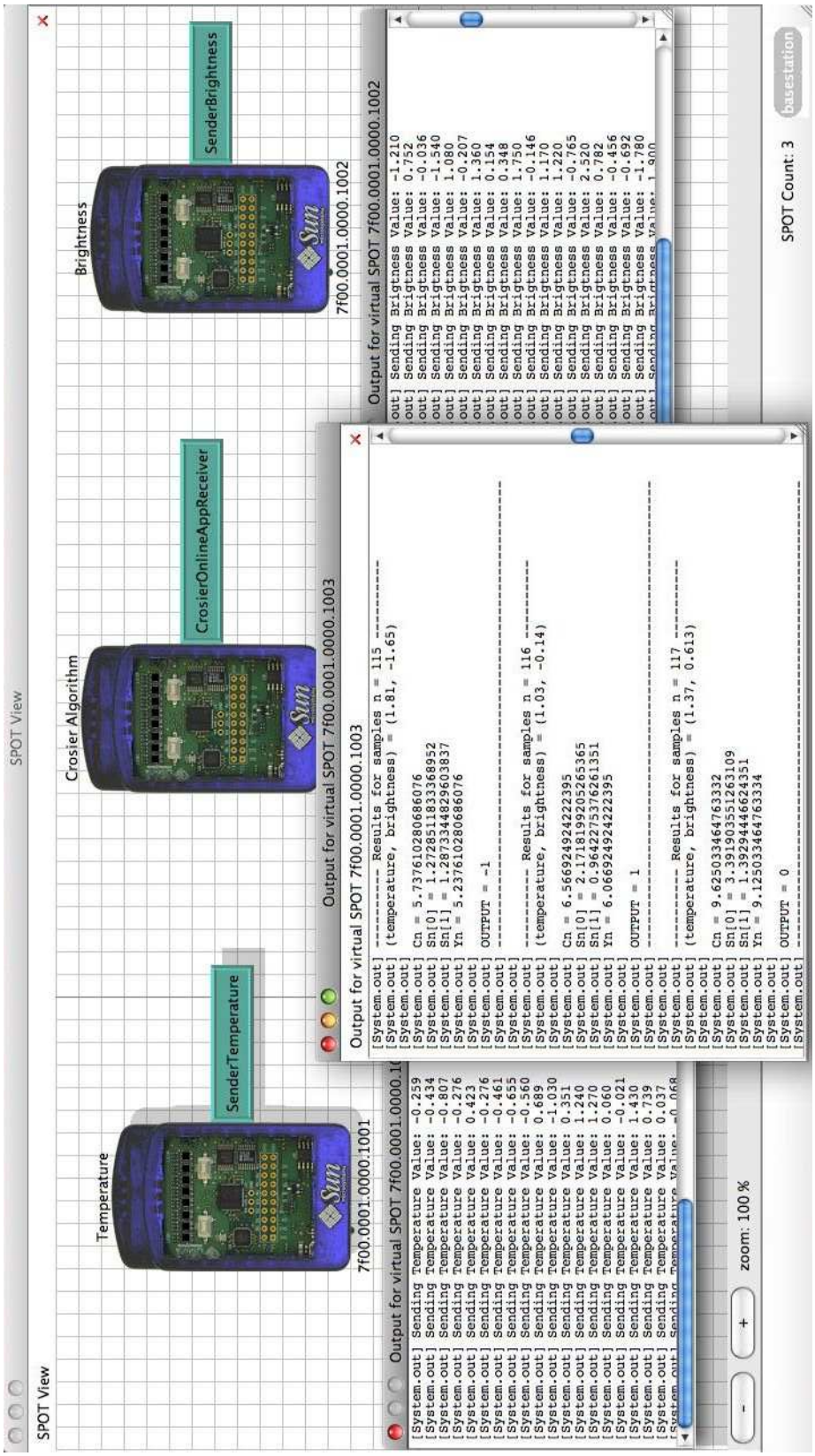
Σχήμα 30: *Ολοκλήρωση της λήψης των επόμενων 50 δειγμάτων (θερμοκρασίας, φωτεινότητας) από τα SunSPOTs, ανανέωση όλων των στατιστικών στοιχείων (μέσες τιμές, διακυμάνσεις, πίνακας συνδιακύμανσης) και καινούρια αποτελέσματα αλγορίθμου Crosier*



Σχήμα 31: *Περίπτωση Συναγερμού - Σύστημα εκτός ελέγχου*



Σχήμα 32: *Επαναφορά Συστήματος στην κατάσταση εντός ελέγχου*



Σχήμα 33: *Περίπτωση που το σύστημα αλλάζει καταστάσεις*

ΒΙΒΛΙΟΓΡΑΦΙΑ

[**1**] M. Basseville, I. V. Nikiforov, "*Detection of Abrupt Changes: Theory and Application*", Prentice-Hall Inc.

[**2**] E. S. Page, "*Controlling the Standard Deviation by Cusum and Warning Lines*", Institute of Statistics, Mimeo Series No.356.

[**3**] R. B. Crosier, "*Multivariate Generalizations of Cumulative Sum Quality-Control Schemes*", Technometrics, August 1988, Vol.30, No.3.

[**4**] B. Mesnil, P. Petitgas, "*Detection of changes in time-series of indicators using CUSUM control charts*", EDP Sciences, IFREMER, IRD 2009.

[**5**] S. L. Spindler, "*Evaluation of some multivariate CUSUM schemes*", Rochester Institute of Technology, 1987.

[**6**] J. J. Pignatiello JR., G. C. Runger, "*Comparisons of Multivariate CUSUM Charts*", Journal of Quality Technology, Vol.22, No.3, July 1990.

[**6**] P. Maravelakis, "*An Investigation of Some Characteristics of Univariate and Multivariate Control Charts*", Ph.D Thesis, Department of Statistics.

[**8**] Sun Labs, "*Sun^{TM} Programmer's Manual*", Release v.6.0 (Yellow), November 2010.

[**9**] "`http://sunspotdev.org/index.html`"

[**10**] "`http://www.itl.nist.gov/div898/handbook/index.htm`"

[**11**] "`https://en.wikipedia.org/wiki/Normal_distribution`"

[**12**] "`https://en.wikipedia.org/wiki/Covariance_matrix`"

[**13**] "`http://www.stat-athens.aueb.gr/~jpan/statistiki-skepsi-II/chapter9.pdf`"

[**14**] Α. Στ. Φουντουλάκη, "*Εμπλουτισμός Στατιστικού Ελέγχου Ποιότητας με Τεχνικές Μηχανικής Μάθησης*", Διδακτορική Διατριβή, Πανεπιστήμιο Πατρών, Πολυτεχνική Σχολή, 2011.

[**15**] Μ. Ν. Βαξεβανίδης, "*Στατιστικός Έλεγχος Παραγωγικής Διαδικασίας*" (σημειώσεις μαθήματος).

[**16**] Δ. Αντζουλάκος, "*Στατιστικός Έλεγχος Ποιότητας*" (σημειώσεις μαθήματος).