PENERAPAN PERAMALAN PERMINTAAN BERBASIS METODE ARIMA DALAM PERENCANAAN PRODUKSI STUDI KASUS DI PT. IGLAS (PERSERO)

TUGAS AKHIR

Disusun Oleh :

ADHI KRESNOYUWONO

NRP. 5201 100 009

PROGRAM STUDI SISTEM INFORMASI
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2006

PENERAPAN PERAMALAN PERMINTAAN BERBASIS METODE ARIMA DALAM PERENCANAAN PRODUKSI STUDI KASUS DI PT. IGLAS (PERSERO)

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan
Memperoleh Gelar Sarjana Komputer
Pada
Program Studi Sistem Informasi
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui/Menyetujui

Dosen Pembimbing I	Dosen Pembimbing II
Prof. Dr. Ir. Arif Djunaidy, M.Sc	Wiwik Anggraeni, S.Si. M.Kom.
NIP. 131 633 403	NIP. 132 296 290

SURABAYA
JULI 2006

KATA PENGANTAR

Puji syukur Alhamdulillah kehadirat Allah SWT atas segala rahmat serta hidayah-Nya, sehingga penulis dapat menyelesaikan Laporan Tugas Akhir ini. Penulisan Laporan ini adalah sebagai salah satu syarat kelulusan pada Program Studi Sistem Informasi Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember Surabaya.

Dalam usaha menyelesaikan penulisan Laporan Akhir ini penulis banyak mendapat bantuan dari berbagai pihak baik secara moral maupun material. Oleh karena itu penulis mengucapkan terima kasih dan penghargaan setinggi-tingginya kepada :

1. Kedua orang tua penulis, eyang, dan bude yang selalu mengingatkan, memberi dorongan dan bantuan, sehingga penulis dapat menyelesaikan tugas akhir ini, sekaligus menyelesaikan pendidikan di Program Studi Sistem Informasi, ITS.
2. Bapak Prof. Dr. Ir. Arif Djunaidy, M.Sc, selaku dosen pembimbing I yang telah banyak memberikan bimbingan dan dukungan kepada penulis hingga laporan tugas akhir ini terselesaikan.
3. Ibu Wiwik Anggraeni, S.Si. M.Kom, selaku dosen pembimbing II yang telah banyak memberikan bimbingan, dukungan moral, motivasi, dan berbagai bantuan lain untuk menyelesaikan tugas akhir ini.
4. Bapak Eko, selaku pembimbing lapangan dari PT. IGLAS (Persero), yang telah memberikan banyak waktu dan tenaga untuk konsultasi dan dukungan data.
5. Ibu Erma Suryani dan Bapak Edwin Riksakomara, selaku dosen penguji tugas akhir, yang telah memberikan koreksi demi kesempurnaan dan juga telah membantu saat pengajuan proposal.
6. Bapak Faisal Johan, selaku dosen penguji tugas akhir yang telah memberikan koreksi untuk tugas akhir ini demi kesempurnaan.
7. Bapak Ir. Khakim Ghozali, selaku Ketua Program Studi Sistem Informasi ITS yang telah memberikan ijinnya untuk ujian tugas akhir.

8. Rekan-rekan seperjuangan di Program Studi Sistem Informasi ITS, IS01, NFORS 2002, SI-Shogun 2003, Angkatan 2004 dan 2005, yang telah sama-sama berjuang mencari ilmu.
9. Seluruh staff dan karyawan Program Studi Sistem Informasi dan Fakultas Teknologi Informasi ITS.
10. Pak Agus Kayanto dari TELKOM, yang sama-sama berjuang untuk tesisnya di MMT ITS dan juga telah banyak memberikan masukan.
11. Teman-temanku yang telah banyak membantu dan memberikan dukungan, Dandy, Rifky, Brian dan lainnya.
12. Berbagai pihak yang belum sempat penulis sebutkan, atas jasa-jasanya dalam membantu dan mendukung penyelesaian tugas akhir ini.

Surabaya, 9 Agustus 2006

Penulis

ABSTRAK

PT. IGLAS (Persero) merupakan sebuah industri manufaktur yang bergerak di bidang pembuatan gelas kemas. Seiring dengan perkembangan permintaan yang semakin dinamis, perlu adanya optimasi pada berbagai sektor, termasuk pula efisiensi terhadap biaya-biya yang terjadi. Biaya lembur dan biaya penyimpanan persediaan adalah beberapa biaya yang biasa terjadi pada perencanaan produksi dan pengendalian persediaan. Pemanfaatan teknik peramalan sebagai salah satu proses pendahuluan dalam melakukan perencanaan produksi diyakini lebih baik daripada hanya mengandalkan intuisi dan pertimbangan subyektif. ARIMA sebagai suatu metode peramalan yang memungkinkan penggabungan konsep model regresi yang dikenal dengan *Autoregression* (AR) dengan metode penghalusan *Moving Average* (MA), dimanfaatkan dalam menyusun perencanaan produksi. Biaya lembur dan biaya penyimpanan persediaan yang terjadi pada proses produksi beberapa produk untuk beberapa periode diminimumkan dengan memanfaatkan salah satu parameternya, yaitu hasil peramalan dan stok pengaman.

Dalam tugas akhir ini akan dikembangkan suatu aplikasi yang dapat melakukan langkah-langkah teknik peramalan dengan metode ARIMA Box-Jenkins, sekaligus menerapkan hasilnya dalam optimasi jumlah biaya-biaya tersebut. Jumlah biaya-biaya tersebut kemudian dibandingkan dengan jumlah biaya-biaya yang dihitung tanpa memanfaatkan hasil ramalan.

Teknik peramalan dengan metode ARIMA memberikan hasil peramalan yang berperan sebagai masukan untuk menghitung total biaya-biaya tersebut. Hasil yang didapat menyatakan total biaya-biaya tersebut menjadi lebih kecil ketika pendekatan teknik peramalan digunakan dibandingkan penggunaan perkiraan subyektif.

Kata kunci : *peramalan, ARIMA, Box-Jenkins, perencanaan produksi, stok pengaman*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SEGMEN PROGRAM

BAB I

PENDAHULUAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pada negara-negara berkembang, sebagian besar perusahaan-perusahaan manufaktur menggunakan intuisi dan pertimbangan secara subyektif ketika membuat perencanaan produksi (*production planning*) dan peramalan permintaan (*demand forecast*). Hal ini merupakan salah satu faktor yang menyebabkan produksi menjadi tidak efisien. Dalam merencanakan produksi diperlukan prediksi jumlah permintaan yang cukup akurat, sehingga biaya yang terjadi dapat diminimumkan.

PT. IGLAS (Persero) merupakan salah satu perusahaan manufaktur (bergerak di bidang pembuatan botol) yang membutuhkan perencanaan produksi yang optimal. Tingkat penjualan produk mereka menjadi semakin dinamis, seiring dengan permintaan pasar yang terjadi. Agar dapat terus bertahan, maka PT. IGLAS (Persero) berusaha mengoptimalkan seluruh kegiatan pada setiap sektornya untuk bekerja seefektif dan seefisien mungkin, termasuk pula pada sektor produksinya.

Salah satu langkah yang dapat diterapkan adalah membuat perencanaan produksi yang memiliki biaya-biaya minimum, termasuk didalamnya biaya lembur (*overtime cost*) dan biaya persediaan (*inventory cost*), namun tetap dapat memenuhi permintaan. Aktifitas peramalan permintaan tentu dibutuhkan dalam hal ini.

Peramalan permintaan merupakan salah satu dari beberapa *critical input* dalam proses perencanaan produksi. Ketika hasil peramalan tidak akurat, rencana produksi menjadi tidak dapat dipercaya, karena akan menimbulkan permasalahan pada persediaan, apakah itu berlebih atau kekurangan. Oleh karena itu, diperlukan stok pengaman (*safety stock*) dalam jumlah yang tepat.

Stok pengaman digunakan untuk mengantisipasi ketidakpastian permintaan relatif terhadap ramalan-ramalan yang dibuat. Kesalahan hasil peramalan inilah yang dijadikan dasar dalam menentukan jumlah stok pengaman, dimana kesalahan hasil peramalan berbanding lurus dengan jumlah stok pengaman. Sedangkan stok pengaman sendiri merupakan salah satu pembatas dalam menentukan total biaya lembur dan biaya persediaan yang terdapat pada perencanaan produksi.

Dengan demikian, semakin kecil kesalahan peramalan yang didapat, maka semakin kecil pula jumlah stok pengamannya. Yang tentunya juga akan memperkecil total biaya lembur dan biaya persediaan. Karenanya diperlukan metode peramalan yang dapat diandalkan untuk berbagai tipe data runtut waktu, yaitu metode ARIMA.

1.2 Tujuan

Titik berat atau tujuan akhir dalam tugas akhir ini adalah melakukan optimasi total biaya lembur dan biaya penyimpanan persediaan beberapa jenis produk untuk beberapa periode pada perencanaan produksi di PT IGLAS (Persero) dengan menerapkan peramalan permintaan dengan metode ARIMA Box-Jenkins. Dan membandingkan hasilnya dengan hasil yang didapat oleh perlakuan yang sama menggunakan prediksi permintaan berdasarkan intuisi.

1.3 Manfaat

Secara garis besar, manfaat yang dapat diperoleh dari tugas akhir ini adalah memberikan solusi sistematik sebagai pembanding dari solusi yang didasarkan atas subyektifitas dan intuisi dalam membuat suatu perencanaan produksi.

1.4 Permasalahan

Masalah-masalah utama yang dititikberatkan pada tugas akhir ini adalah :

a. Bagaimana implementasi langkah-langkah teknik peramalan dengan metode ARIMA Box-Jenkins dilakukan pada aplikasi yang dibangun.
b. Bagaimana perbandingan antara total biaya lembur dan penyimpanan persediaan pada perencanaan produksi yang hasil perhitungannya memanfaatkan peramalan model ARIMA dengan hasil yang perhitungannya berdasarkan intuisi (subyektifitas).

1.5 Batasan Masalah

a. Metode peramalan yang digunakan adalah ARIMA Box-Jenkins Non-Seasonal.
b. Tingkat *p* dan *q* yang berturut-turut merupakan order dari *autoregressive* (AR) dan *moving average* (MA) mengacu pada model ARIMA yang umum yaitu antara 0, 1, atau 2.
c. Data yang diambil dianggap bukan merupakan data musiman (komponen / sifat musiman bukan yang dominan).
d. Perencanaan produksi yang dimaksud dibatasi pada perencanaan total biaya lembur dan biaya penyimpanan persediaan.
e. Permintaan diasumsikan periodik dan di luar kejadian-kejadian luar biasa.
f. Nilai parameter-parameter yang digunakan dalam perencanaan produksi diasumsikan tidak berubah.
g. Lingkungan pemrograman yang digunakan untuk mengembangkan aplikasi adalah Java dan dijalankan pada sistem operasi Windows.

1.6 Metodologi

Pengerjaan tugas akhir ini meliputi beberapa tahapan, berikut adalah tahap-tahap yang akan dilakukan :

a. Studi Kepustakaan

Tahap ini dilakukan untuk mempelajari konsep, teori-teori tentang teknik peramalan dengan metode ARIMA, serta mempelajari bagaimana optimasi total biaya lembur dan biaya penyimpanan dapat dilakukan.

b. Pengumpulan Data

Pada tahap ini dilakukan pengumpulan data, dengan berbagai metode yang mungkin (wawancara, observasi, dan sebagainya), baik yang berasal dari perusahaan maupun data pelengkap lain. Data yang dikumpulkan berupa data permintaan yang telah lalu di perusahaan tersebut

c. Pemodelan Sistem

Setelah dapat melakukan peramalan dan dicapai hasil yang cukup layak, kemudian dilakukan pemodelan sistem untuk mengoptimasi total biaya lembur dan biaya penyimpanan persediaan. Sistem ini akan mendasari tahap selanjutnya

d. Desain Aplikasi

Pada tahap ini dilakukan desain data maupun desain antarmuka aplikasi. Dilanjutkan dengan pengembangan aplikasi

e. Implementasi Aplikasi

Setelah melakukan desain data maupun antarmuka aplikasi, aplikasi dikembangkan dengan menyesuaikan berdasarkan desain yang telah dibuat. Aplikasi dikembangkan dengan bahasa pemrograman Java

f. Uji Coba dan Analisa

Menguji aplikasi yang telah jadi dengan variasi masukan sehimpunan data, kemudian melakukan analisa terhadap hasil yang dicapai

g. Penyusunan Buku Tugas Akhir

Pada tahap terakhir ini akan disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir. Dokumen ini diharapkan dapat berguna bagi pembaca yang memiliki keinginan untuk mengkaji lebih lanjut atau untuk keperluan pengembangan sistem menuju ke arah kesempurnaan

1.7 Sistematika Penulisan Buku Tugas Akhir

Penulisan buku tugas akhir sebagai laporan pelaksanaan tugas akhir disusun dengan sistematika sebagai berikut :

Bab I merupakan bagian Pendahuluan. Dalam bab ini dibahas hal-hal yang berhubungan dengan rencana pelaksanaan tugas akhir ini. Berisi latar belakang, tujuan, manfaat, perumusan masalah, pembatasan dan asumsi masalah, metodologi, dan sistematika penulisan laporan.

Bab II merupakan bagian Dasar Teori. Dalam bab ini dibahas teori-teori yang berkaitan dengan metode peramalan ARIMA, termasuk diantaranya identifikasi model, estimasi parameter, diagnosis, dan dibahas pula mengenai pemrograman linier. Termasuk pula dasar teori mengenai klasifikasi data runtut waktu.

Bab III merupakan bagian Analisis dan Perencanaan Produksi di PT. IGLAS (Persero). Dalam bab ini dibahas hal-hal yang berkaitan dengan aktifitas perencanaan produksi yang dihadapi PT. IGLAS (Persero). Termasuk identifikasi variabel keputusan dan parameter perencanaan produksi.

Bab IV merupakan bagian Analisis dan Desain Aplikasi. Dalam bab ini dibahas mengenai analisis dan desain aplikasi yang ditujukan sebagai perangkat lunak yang merepresentasikan penerapan metode ARIMA dalam meramalkan jumlah permintaan yang berperan sebagai masukan dalam perencanaan produksi sehingga hasil yang dicapai dapat optimal. Bahasa pemodelan visual UML digunakan di sini.

Bab V merupakan bagian yang membahas Implementasi Aplikasi. Dalam bab ini dibahas penerapan hasil analisis dan desain aplikasi berkaitan dengan bahasa dan algoritma pemrograman serta penggunaan struktur data dalam membangun aplikasi.

Bab VI merupakan bagian Uji Coba dan Analisis Data. Dalam bab ini diberikan beberapa skenario-skenario uji coba, yang diikuti dengan analisis terhadap hasil uji coba tersebut.

Bab VII merupakan Penutup. Bagian ini memberikan kesimpulan dari pelaksanaan tugas akhir ini, diikuti dengan saran untuk pengembangan selanjutnya.

BAB II

DASAR TEORI

BAB II

DASAR TEORI

Dalam bab ini dijelaskan beberapa dasar teori yang berkaitan dengan metode peramalan ARIMA Box-Jenkins dan pemrograman linier. Penjelasan dimulai dengan klasifikasi data secara umum dan sedikit uraian mengenai data runtut waktu (*time series*). Penjelasan dilanjutkan dengan teori-teori mengenai beberapa hal yang berkaitan metode peramalan ARIMA Box-Jenkins, antara lain : tahap-tahap pendekatan Box-Jenkins, fungsi autokorelasi dan fungsi autokorelasi parsial, proses pengujian dan pengecekan diagnostik, dan model-model ARIMA. Kemudian dilanjutkan dengan uraian penjelasan mengenai pemrograman linier.

2.1 Klasifikasi Data

Unsur utama yang berkaitan dengan kegiatan peramalan adalah data. Data merupakan fakta, yaitu sesuatu yang keberadaannya dapat diketahui melalui panca indera. Bila data diolah sehingga memiliki arti, maka data itu dinamakan informasi [LER-02]. Data dapat diklasifikasikan berdasarkan wujudnya, sumbernya, maupun dimensi waktunya.

Berdasarkan wujudnya, data dibedakan menjadi data kuantitatif dan data kualitatif. Data yang berwujud bilangan dinamakan data kuantitatif, sedangkan data yang tidak berwujud bilangan dinamakan data kualitatif. Contoh data kuantitatif yaitu jumlah penjualan, jumlah pesanan, harga bahan bakar. Sedangkan contoh data kualitatif yaitu data mengenai kondisi sosial politik, data mengenai karyawan perusahaan yang resah.

Berdasarkan sumbernya, data dibedakan menjadi data internal dan data eksternal. Data internal yaitu data yang berasal dari dalam organisasi (perusahaan), sedangkan data yang berasal dari luar organisasi (perusahaan) disebut data eksternal.

Berdasarkan dimensi waktunya, data dibedakan menjadi data runtut waktu (*time series*) dan data *cross sectional*. Data runtut waktu merupakan data yang dikumpulkan dari suatu waktu ke waktu berikutnya selama jangka waktu tertentu. Misalnya, data mengenai jumlah pesanan selama dua belas bulan dalam jangka waktu satu tahun. Data *cross sectional* merupakan data yang dikumpulkan pada satu waktu tertentu, tanpa memiliki variasi dimensi waktu. Misalnya, data mengenai jumlah penjualan beberapa perusahaan pada tahun 1998. Untuk konteks peramalan, data yang dianggap relevan adalah data runtut waktu.

2.2 Data Runtut Waktu

Data runtut waktu dapat dibedakan menjadi data runtut waktu yang bersifat stasioner dan tidak stasioner. Data stasioner relatif tetap dari waktu ke waktu sedangkan data tidak stasioner relatif bervariasi dari waktu ke waktu.

Data runtut waktu dapat juga dibedakan menjadi empat komponen, yaitu trend, musim, siklis, dan ketidak-teraturan (*iregular*) atau acak (*random*). Data runtut waktu mungkin saja terdiri atas keempat komponen itu sekaligus. Trend merupakan komponen data runtut waktu yang berkaitan dengan adanya kecenderungan (meningkat atau menurun) dalam jangka panjang. Musim merupakan komponen data runtut waktu yang berkaitan dengan adanya kejadian yang berulang secara teratur dalam setiap waktu. Siklis merupakan komponen data runtut waktu yang berkaitan dengan adanya kejadian yang tidak teratur. Komponen ini terjadi dalam kurun waktu yang lebih dari satu tahun dan biasanya dengan periode yang tidak sama. Yang terakhir adalah acak. Yaitu komponen data runtut waktu yang tidak tergolong dalam trend, musim, maupun siklis. Pola data terjadi secara tidak sistematis, sehingga komponen ini sering juga disebut komponen acak, *iregular*, atau *error*.

Fungsi variabel yang digunakan, dibedakan menjadi variabel *dependen* dan variabel *independen*. Variabel *dependen*, merupakan variabel yang keberadaannya akan diramalkan atau dijelaskan pada waktu yang akan datang. Sedangkan varibel *independen*, merupakan variabel yang digunakan untuk

meramalkan atau menjelaskan keberadaan variabel *dependen* pada waktu yang akan datang.

2.3 Metode untuk Menganalisa Data Runtut Waktu

Terdapat beberapa metode untuk menganalisa data runtut waktu. Dari analisa itu dapat diperoleh informasi mengenai bagaimana pola suatu data runtut waktu. Sebagian metode untuk menganalisa data runtut waktu antara lain yaitu plot data dan analisa koefisien autokorelasi serta analisa koefisien autokorelasi parsial.

2.3.1 Plot Data

Salah satu metode yang cukup baik untuk menganalisis data runtut waktu adalah dengan memplot data tersebut secara grafis. Sebagai contoh, dengan plot data maka dapat diketahui ada tidaknya trend (penyimpangan nilai rata-rata) dengan membandingkan antara plot berbagai versi data *moving average*, dengan plot data *moving average* empat periode maka pengaruh musim pada data dapat dihilangkan, dan sebagainya.

2.3.2 Analisa Koefisien Autokorelasi dan Autokorelasi Parsial

Korelasi antara dua variabel menjelaskan hubungan antara kedua variabel itu. Nilai korelasi merepresentasikan tingkat pengaruh antara perubahan satu variabel dengan perubahan variabel lainnya.

Tingkat korelasi ini diukur dengan koefisien korelasi yang besarnya bervariasi di antara +1 dan –1. Suatu nilai koefisien korelasi yang mendekati +1 menunjukkan kuatnya hubungan positif diantara dua variabel itu. Ini berarti bahwa bila nilai dari salah satu variabel meningkat atau bertambah, maka nilai variabel lainnya juga cenderung bertambah. Demikian pula halnya dengan nilai koefisien korelasi yang mendekati –1, menunjukkan bertambahnya nilai salah satu variabel akan mengakibatkan turunnya atau kurangnya nilai dari variabel lainnya. Suatu nilai koefisien korelasi nol menunjukkan bahwa kedua variabel secara

statistik adalah bebas, tidak tergantung satu dengan lainnya, sehingga tidak ada perubahan dalam satu variabel, bila variabel lainnya berubah.

Autokorelasi adalah hubungan antara variabel dengan lag waktu satu atau lebih dengan variabel itu sendiri [HAN-01]. Dengan melakukan analisa autokorelasi, pola dari suatu data runtut waktu dapat diketahui. Besarnya autokorelasi biasa disebut dengan koefisien autokorelasi, yang dapat diperoleh dengan persamaan (2.1) berikut [MAK-98] :

$$r_k = \frac{\sum_{t=1}^{n-k}(Y_t - \bar{Y})(Y_{t+k} - \bar{Y})}{\sum_{t=1}^{n}(Y_t - \bar{Y})^2} \qquad (2.1)$$

dimana :

r_k = koefisien autokorelasi untuk lag waktu k ;

Y_t = data dari *series* ;

$\bar{Y}$ = rata-rata dari *series* ;

Y_{t+k} = data dengan k periode di depan.

Setelah besar koefisien autokorelasi untuk tiap lag (lag 1, lag 2, lag 3, dan seterusnya) dihitung dengan persamaan (2.1), kemudian nilai-nilainya dapat digambarkan dalam grafik yang biasa disebut dengan *correlogram*. Atau dengan kata lain, *correlogram* adalah grafik dari autokorelasi untuk berbagai lag data runtut waktu [HAN-01].

Beberapa hal yang berkaitan dengan pola data, dapat dipelajari dari koefisien autokorelasi maupun *correlogram*. Secara teori, data dikatakan acak, jika autokorelasi antara Y_t dengan Y_{t-k} untuk setiap lag mendekati nol, sehingga nilai-nilai dalam data runtut waktu tersebut tidak berhubungan satu sama lain. Data dikatakan memiliki komponen trend, jika koefisien autokorelasinya secara signifikan berbeda dari nol untuk beberapa lag awal dan secara perlahan menuju ke nol seiring dengan pertambahan periode. Data dikatakan memiliki pola

musiman, jika koefisien autokorelasinya bernilai signifikan pada lag waktu musiman, misal data caturwulanan (*quarterly*) signifikan pada lag 4, 8, 12, ...

Selain autokorelasi dikenal pula autokorelasi parsial. Autokorelasi parsial digunakan untuk mengukur keeratan antara Y_t dan Y_{t-k} apabila pengaruh dari lag waktu 1, 2, 3, ..., k –1 dianggap terpisah atau dihilangkan. Nilai koefisien autokorelasi parsial dapat diperoleh dengan persamaan (2.2) dan (2.3) berikut [NGU-02] :

$$r_{kk} = \frac{r_k - \sum_{j=1}^{k-1}(r_{k-1,j})(r_{k-j})}{1 - \sum_{j=1}^{k-1}(r_{k-1,j})(r_j)} \tag{2.2}$$

$$r_{kj} = r_{k-1,j} - (r_{kk})(r_{k-1,k-j}) \tag{2.3}$$

dimana :

r_k = koefisien autokorelasi dengan lag k ;

r_{kj} = koefisien autokorelasi parsial untuk lag k , ketika efek dari lag yang berjumlah j telah dihilangkan, dengan nilai j memiliki nilai dari 1 sampai $k - 1$.

Fungsi autokorelasi (ACF) dan fungsi autokorelasi parsial (PACF) digunakan untuk identifikasi pada model ARIMA Box-Jenkins.

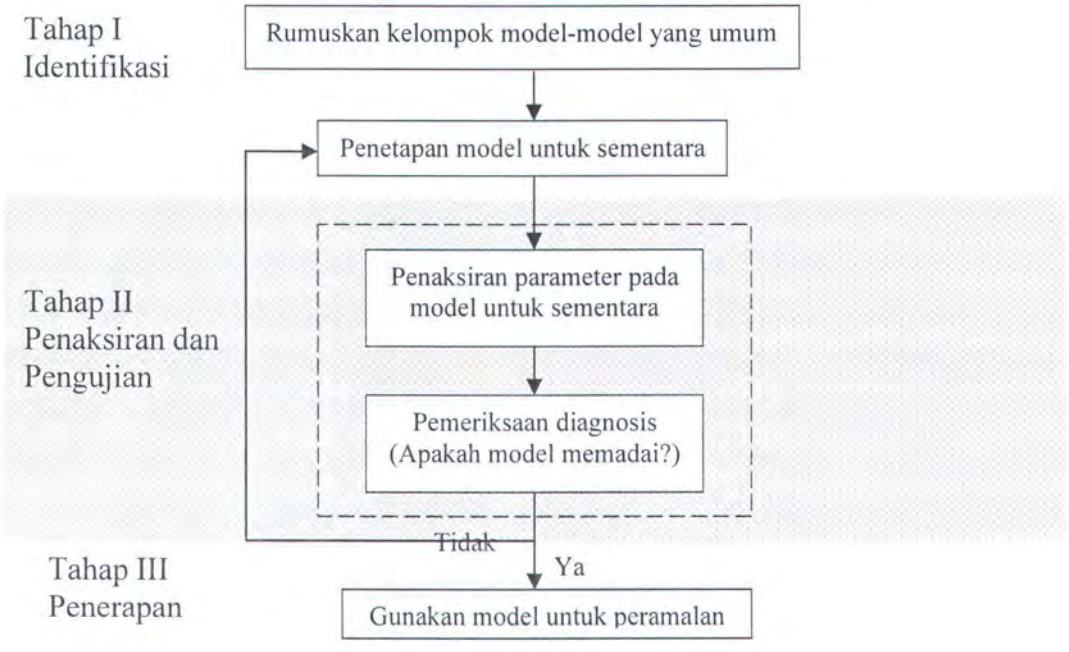
2.4 Metode Peramalan ARIMA Box–Jenkins

Metode peramalan *Autoregressive Integrated Moving Average* (ARIMA) dikembangkan oleh Box dan Jenkins sehingga disebut ARIMA Box-Jenkins. Metodologi Box-Jenkins merupakan serangkaian prosedur untuk identifikasi, pencocokan, dan pemeriksaan model ARIMA dengan data runtut waktu. Analisis yang digunakan oleh teknik peramalan ini hanya menggunakan satu variabel *dependen*, yang selanjutnya dikembangkan menjadi beberapa variabel *independen*. Peramalan akan mengikuti bentuk model yang cocok secara langsung [HAN-01].

Data runtut waktu yang dianalisis pada metode ARIMA adalah data yang stasioner. Apabila data telah stasioner, maka data tersebut selanjutnya dapat diidentifikasi dan kemudian mengikuti tahap-tahap pendekatan Box-Jenkins. Namun apabila data belum stasioner, maka diperlukan proses *differencing* untuk memperoleh data runtut waktu yang stasioner.

2.4.1 Tahap-Tahap Pendekatan Box-Jenkins

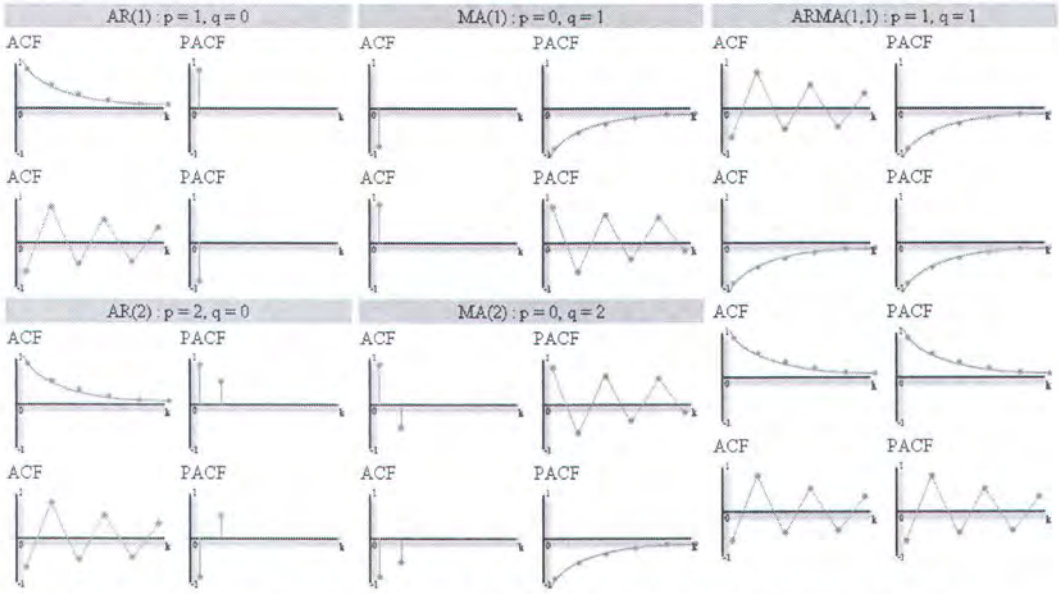
Untuk menggambarkan peramalan dengan pendekatan Box-Jenkins, maka George Box dan Gwilyn Jenkins telah mengembangkan suatu diagram skema, seperti digambarkan pada gambar 2.1 [MAK-98]. Skema ini terbagi dalam tiga tahap, yaitu identifikasi, penaksiran dan pengujian, dan penerapan. Pada tahap pertama, suatu model tertentu dimasukkan sementara sebagai model yang dianggap cocok untuk keadaan yang teridentifikasi. Tahap kedua mencocokkan model tersebut dengan data historis yang tersedia dan melakukan pengecekan untuk menentukan apakah model tersebut sudah cukup tepat. Jika tidak tepat, maka kembali ke tahap pertama, namun jika sudah cukup tepat, maka tahap ketiga dilakukan, yaitu penyusunan ramalan untuk periode yang akan datang.



Gambar 2.1 Skema Pendekatan Box-Jenkins

2.4.2 Model-model untuk Data Runtut Waktu

Data runtut waktu yang telah stasioner selanjutnya dapat dimodelkan sebagai model ARIMA. Model-model tersebut antara lain yaitu model *Autoregressive* (AR), model *Moving Average* (MA), dan model *Autoregressive Moving Average* (ARMA). Model-model ARIMA dapat identifikasi dari fungsi autokorelasi (ACF) dan fungsi autokorelasi parsial (PACF) data runtut waktu. Gambar 2.2 menunjukkan *correlogram* untuk ACF dan PACF data runtut waktu, yang mengidentifikasikan model-model umum ARIMA [HAN-01].



Gambar 2.2 Correlogram untuk ACF dan PACF Model-model ARIMA yang Umum

2.4.2.1 Model Autoregressive (AR)

Model *autoregressive* memiliki bentuk persamaan umum seperti pada persamaan (2.4) berikut [HAN-01]:

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \varepsilon_t \qquad (2.4)$$

dimana :

Y_t = Variabel yang diamati pada waktu t

$Y_{t-1}, Y_{t-2}, ..., Y_{t-p}$ = Variabel yang diamati pada waktu t-1, t-2, …, t-p.

$\phi_0, \phi_1, \phi_2, ..., \phi_p$ = Koefisien yang harus diestimasi.

ε_t = Kesalahan (*Error*) pada saat t

Jumlah pengamatan stasioner masa lalu yang digunakan model *autoregressive* disebut dengan tingkat p. Sehingga jika digunakan dua pengamatan masa lalu ($p = 2$), maka model disebut dengan AR(2), dan dinyatakan dalam persamaan :

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon_t \qquad (2.5)$$

Model *autoregressive* harus memenuhi syarat tertentu, yaitu jumlah koefisien *autoregressive* harus lebih kecil dari satu, atau $\phi_1 + \phi_2 + ... + \phi_p < 1$. Syarat ini disebut kondisi *stationary* [NGU-02].

2.4.2.2 Model Moving Average (MA)

Model *moving average* memiliki bentuk persamaan umum seperti pada persamaan (2.6) berikut [HAN-01]:

$$Y_t = \mu + \varepsilon_t - \omega_1 \varepsilon_{t-1} - \omega_2 \varepsilon_{t-2} - ... - \omega_q \varepsilon_{t-q} \qquad (2.6)$$

dimana :

Y_t	= Variabel yang diamati pada waktu t
μ	= Konstanta mean
$\omega_1, \omega_2, ..., \omega_q$	= Koefisien yang diestimasi
ε_t	= Kesalahan (*Error*)
$\varepsilon_{t-1}, \varepsilon_{t-2}, ..., \varepsilon_{t-q}$	= Kesalahan (*Error*) masa lalu

Jumlah kesalahan (*error*) masa lalu yang digunakan dalam model *moving average* disebut sebagai tingkat q. Sehingga jika digunakan dua kesalahan masa lalu ($q = 2$), maka model itu disebut dengan MA(2), dan dinyatakan dalam persamaan :

$$Y_t = \mu + \varepsilon_t - \omega_1 \varepsilon_{t-1} - \omega_2 \varepsilon_{t-2} \qquad (2.7)$$

Ada beberapa batasan yang harus dipenuhi dalam model *moving average* ini. Syarat penting (tapi tidak cukup) yang harus dipenuhi adalah jumlah dari

semua koefisien moving average harus lebih kecil dari satu, atau $\omega_1 + \omega_2 + ... + \omega_q < 1$. Syarat ini disebut sebagai kondisi *invertibility* [NGU-02].

2.4.2.3 Model Autoregressive Moving Average (ARMA)

Model *Autoregressive Moving Average* (ARMA) merupakan paduan dari model *autoregressive* (AR) dengan model *moving average* (MA). Notasi yang biasa digunakan adalah ARMA(p, q), dimana p merupakan tingkat dari bagian autoregressive dan q merupakan tingkat dari bagian moving average. Bentuk persamaan umum dari model ini adalah seperti pada persamaan (2.8) berikut [HAN-01] :

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \varepsilon_t - \omega_1 \varepsilon_{t-1} - \omega_2 \varepsilon_{t-2} - ... - \omega_q \varepsilon_{t-q} \quad (2.8)$$

Model ARMA(p, q) dapat menjelaskan berbagai variasi pola data runtut waktu yang stasioner. Sebagai contoh, ARMA(1, 0) sama saja dengan AR(1), dan ARMA(0, 1) sama saja dengan MA(1).

Apabila suatu data runtut waktu belum stasioner, maka diperlukan proses *differencing*, seperti dijelaskan sebelumnya. Tingkat *differencing* terhadap data runtut waktu tersebut adalah d. Dan model untuk data runtut waktu tersebut adalah ARIMA(p, d, q).

Jika model ARMA(p, q) dapat menjelaskan variasi data runtut waktu yang stasioner, maka ARIMA(p, d, q) dapat menjelaskan variasi data runtut waktu baik yang stasioner ($d = 0$) maupun yang tidak stasioner($d > 0$).

Pada prakteknya, jarang diperlukan pemakaian nilai p, d, atau q selain dari 0, 1, atau 2 [MAK-98]. Sehingga, pemakaian nilai p, d, atau q yang berkisar antara 0, 1, atau 2 diharapkan dapat mencakup situasi peramalan praktis.

2.4.3 Penaksiran Parameter Model

Setelah model dari data teridentifikasi, langkah selanjutnya adalah penaksiran parameter model. Penaksiran (estimasi) parameter model ditujukan untuk mendapatkan nilai-nilai parameter yang meminimumkan jumlah kuadrat nilai sisa atau *sum of squared residuals* (SSR).

2.4.3.1 Penaksiran Parameter Model AR(1)

Pada model AR(1) terdapat dua parameter yang akan diestimasi yaitu konstanta ϕ_0 dan koefisien ϕ_1. Persamaan (2.9) dan (2.10) akan menemukan nilai estimasi untuk kedua parameter tersebut.

$$\phi_1 = \frac{n\sum_{t=1}^{n} y_{t-1}y_t - (\sum_{t=1}^{n} y_{t-1})(\sum_{t=1}^{n} y_t)}{n\sum_{t=1}^{n} (y_{t-1})^2 - (\sum_{t=1}^{n} y_{t-1})^2} \tag{2.9}$$

$$\phi_0 = \frac{\sum_{t=1}^{n} y_t}{n} - \phi_1 \frac{\sum_{t=1}^{n} y_{t-1}}{n} \tag{2.10}$$

Sebenarnya persamaan (2.9) dan (2.10) tersebut sama dengan cara penyelesaian koefisien untuk regresi linier sederhana dengan satu variabel *independen* (bebas), hanya saja dalam hal ini yang berperan sebagai variabel *independen* adalah sama dengan variabel *dependen* (terikat)nya (Y_t) namun pada satu periode sebelumnya (Y_{t-1}).

Nilai koefisien *autoregressive*, ϕ_1, hasil estimasi yang telah diperoleh harus memenuhi persyaratan kondisi $-1 < \phi_1 < 1$. Jika tidak, bisa jadi model yang diajukan tidak sesuai.

2.4.3.2 Penaksiran Parameter Model AR(2)

Terdapat tiga parameter yang akan diestimasi yaitu konstanta ϕ_0 dan koefisien-koefisien ϕ_1 dan ϕ_2. Untuk menemukan nilai estimasi ketiga parameter tersebut digunakan cara penyelesaian yang sama dengan regresi linier berganda yaitu menggunakan metode dekomposisi Choleski. Metode ini menggunakan aljabar matriks dalam komputasi koefisiennya.

Langkah awal sebelum melakukan metode ini adalah membentuk suatu matriks dari himpunan nilai-nilai data serta transpose dari matriks tersebut.

$$X = \begin{bmatrix} 1 & x_{21} & x_{31} \\ 1 & x_{22} & x_{32} \\ 1 & x_{23} & x_{33} \\ : & : & : \\ 1 & x_{2n} & x_{3n} \end{bmatrix} \quad X^T = \begin{bmatrix} 1 & 1 & 1 & .. & 1 \\ x_{21} & x_{22} & x_{23} & .. & x_{2n} \\ x_{31} & x_{32} & x_{33} & .. & x_{3n} \end{bmatrix} \qquad (2.11)$$

Matriks di atas dibentuk berdasarkan persamaan regresi linier berganda dengan variabel *independen* x_1 dan x_2 untuk periode ke 1 sampai periode ke n. Untuk model AR(2) yang memiliki persamaan *autoregressive* dengan variabel *independen* y_{t-1} dan y_{t-2} , maka nilai data pada matriks tersebut diganti dengan nilai-nilai data untuk variabel y_{t-1} dan y_{t-2}. Langkah selanjutnya adalah melakukan operasi perkalian pada X^T dengan X, yang kemudian matriks hasil perkalian itu ($X^T X$) didekomposisi menjadi LL^T, dimana L adalah matriks segitiga bawah.

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{12} & l_{22} & 0 \\ l_{13} & l_{23} & l_{33} \end{bmatrix} \quad L^T = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix} \qquad (2.12)$$

Dengan beberapa langkah perhitungan didapatkan persamaan (2.13) [MAK,1998].

$$LL^T b = X^T Y \qquad (2.13)$$

Matriks *b* adalah matriks berordo 3x1 yang memuat koefisien regresi. Jika dimisalkan $L^T b = Z$, maka persamaan (2.13) menjadi

$$LZ = X^T Y \qquad (2.14)$$

Dari persamaan (2.14) akan diperoleh tiga buah persamaan linier, dan jika dilakukan subtitusi , maka nilai elemen-elemen *Z* dapat diketahui. Setelah nilai elemen-elemen *Z* diketahui, maka nilai elemen-elemen *b* akhirnya juga dapat diketahui.

Sama dengan model AR(1), koefisien model AR(2) yang diperoleh harus memenuhi kondisi-kondisi tertentu, yaitu $-1 < \phi_2 < 1$, $\phi_2 + \phi_1 < 1$, dan $\phi_2 - \phi_1 < 1$.

2.4.3.3 Penaksiran Parameter Model MA(1)

Estimasi parameter untuk model MA(1) dilakukan dengan iterasi yang mencari nilai minimum dari SSR. Iterasi yang dilakukan menyesuaikan pada rentang $-1 < \omega_1 < 1$ menggunakan *step* dengan nilai tertentu untuk mencari nilai ω_1 yang mampu menghasilkan nilai SSR minimum. Setelah nilai ω_1 yang dimaksud ditemukan, maka iterasi dilanjutkan lagi pada rentang di sekitar nilai tersebut, menggunakan nilai *step* yang lebih kecil (lebih teliti), sehingga ditemukan lagi nilai ω_1 yang mampu menghasilkan nilai SSR minimum pada rentang itu, dan tentunya nilai SSR minimum yang ditemukan pada iterasi ini lebih kecil dari nilai minimum SSR sebelumnya. Iterasi ini dilanjutkan lagi pada rentang di sekitar nilai minimum SSR terakhir dengan nilai *step* yang lebih kecil lagi, begitu seterusnya, iterasi dilanjutkan hingga pada nilai *step* tertentu (kedalaman tertentu) yang ditentukan.

Sedangkan untuk konstanta MA lainnya, μ (konstanta mean), merupakan nilai rata-rata data observasi. Sehingga untuk mencari nilai konstanta ini tidak memerlukan iterasi.

2.4.3.4 Penaksiran Parameter Model MA(2)

Langkah-langkah estimasi parameter untuk model MA(2) sama halnya dengan langkah-langkah estimasi parameter untuk model MA(1), namun karena pada model MA(2) koefisien yang digunakan adalah ω_1 dan ω_2, maka untuk mencari nilai SSR minimum iterasinya mempertimbangkan hasil kombinasi nilai kedua koefisien ini. Persyaratan yang harus dipenuhi oleh kedua parameter ini adalah $-1 < \omega_2 < 1$, $\omega_2 + \omega_1 < 1$, dan $\omega_2 - \omega_1 < 1$.

2.4.4 Pengujian dan Pengecekan Diagnostik

Pada tahap-tahap pendekatan Box-Jenkins (gambar 2.1), setelah dilakukan penaksiran (estimasi) parameter. Kemudian dilakukan pengujian dan pengecekan

diagnosa. Untuk melakukan pengujian dan pengecekan diagnostik digunakan *Q-Statistic*.

Q-Statistic digunakan untuk menguji apakah fungsi autokorelasi kesalahan semuanya tidak berbeda dari nol. Rumusan dari *Q-Statistic* ini adalah sebagai berikut [MAK-98] :

$$Q = n(n+2)\sum \frac{r_k^{\,2}}{n-k} \tag{2.15}$$

dimana :

r_k = koefisien autokorelasi kesalahan dengan *lag k*.

n = banyaknya observasi series stasioner.

Q-Statistic mendekati distribusi *chi-square* dengan derajat bebas $k - p - q$. Jika Q-*Statistic* lebih kecil dari nilai kritis *chi-square* seperti yang tertera pada tabel, maka semua koefisien autokorelasi dianggap tidak berbeda dengan nol atau model telah dispesifikasikan dengan benar [NGU-02].

2.5 Pemrograman Linier

Pemrograman linier (*Linear Programming*) adalah suatu cara untuk menyelesaikan persoalan pengalokasian sumber-sumber yang terbatas diantara beberapa aktivitas yang bersaing, dengan cara terbaik yang dilakukan.

Persoalan pengalokasian ini akan muncul manakala seseorang harus memilih tingkat aktivitas-aktivitas tertentu yang bersaing dalam hal penggunaan sumber daya langka yang dibutuhkan untuk melaksanakan aktivitas-aktivitas tersebut.

Pemrograman linier menggunakan model matematis untuk menjelaskan persoalan yang dihadapinya. Sifat linier memberi arti bahwa seluruh fungsi matematis dalam model ini merupakan fungsi yang linier, yang berarti seluruh variabel yang digunakan memiliki pangkat tertinggi 1.

Teknik pemrograman linier telah digunakan meluas di berbagai bidang antara lain perindustrian, pertanian, transportasi, ekonomi, kesehatan, maupun militer.

2.5.1 Komponen-komponen Dasar

Model pemrograman linier, seperti juga model riset operasional lainnya, memiliki tiga komponen dasar, yaitu :

a. Variabel Keputusan (*Decision Variable*)

Merupakan variabel yang akan dicari nilainya, sehingga tercapai hasil optimum pada fungsi tujuan model. Variabel keputusan pada umumnya berharga nonnegatif (*nonnegativity restriction*).

b. Fungsi Tujuan (*Objective / goal*)

Merupakan hasil akhir optimum yang hendak dicapai oleh model, yaitu nilai maksimum (umumnya untuk pendapatan) atau minimum (umumnya untuk biaya).

c. Pembatas-pembatas (*Constraints*)

Merupakan aturan-aturan yang harus dipenuhi oleh model dalam menentukan nilai variable-variabel keputusannya

Pemrograman linier memiliki bentuk umum seperti bentuk-bentuk riset operasional lainnya. Bentuk umum dari model pemrograman linier didasarkan pada ketiga komponen tersebut, yaitu diawali dengan pendefinisian fungsi tujuan (maksimasi atau minimasi), diikuti dengan mendaftarkan setiap pembatasnya.

Penyelesaian permasalahan pemrograman linier dapat dilakukan secara *graphical* maupun secara aljabar. Metode simpleks adalah bentuk penyelesaian secar aljabar.

2.5.2 Metode Simpleks

Merupakan prosedur pemecahan permasalahan pemrograman linier yang bersifat iteratif. Pada metode ini, *constraint* (pembatas) yang biasanya berupa pertidaksamaan diubah dengan persamaan. Dimana tanda ($\leq$) pada *constraint* menyebabkan sisa *resource* atau *slack*, sedangkan tanda ($\geq$) merupakan batas

bawah (minimum) sehingga ada *surplus*. Sebagai ilustrasi, digunakanlah model untuk kasus Reddy Mikks [HAM-03] sebagai berikut :

$$Maksimasi\ z = 5x_1 + 4x_2$$

berdasarkan :

$$\begin{aligned} 6x_1 + 4x_2 &\le 24 \\ x_1 + 2x_2 &\le 6 \\ -x_1 + x_2 &\le 1 \\ x_2 &\le 2 \\ x_1, x_2 &\ge 0 \end{aligned} \qquad (2.16)$$

Model (2.16) tersebut diubah menjadi bentuk standar (2.17) sebagai berikut :

$$Maksimasi\ z = 5x_1 + 4x_2 + 0s_1 + 0s_2 + 0s_3 + 0s_4$$

berdasarkan :

$$\begin{aligned} 6x_1 + 4x_2 + s_1 &= 24 \\ x_1 + 2x_2 + s_2 &= 6 \\ -x_1 + x_2 + s_3 &= 1 \\ x_2 + s_4 &= 2 \\ x_1, x_2, s_1, s_2, s_3, s_4 &\ge 0 \end{aligned} \qquad (2.17)$$

Variabel s_1, s_2, s_3,dan s_4 adalah *slack*, yang berhubungan dengan tanda ($\le$) pada *constraint*.

Langkah selanjutnya adalah merepresentasikan model bentuk standar tersebut ke dalam tabel simpleks. Terlebih dahulu, fungsi tujuan diubah menjadi *objective equation* sebagai berikut :

$$z - 5x_1 - 4x_2 = 0$$

Tabel simpleks mula-mula ditunjukkan sebagai berikut :

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	Solution	
z	1	-5	-4	0	0	0	0	0	baris z
s_1	0	6	4	1	0	0	0	24	baris s_1
s_2	0	1	2	0	1	0	0	6	baris s_2
s_3	0	-1	1	0	0	1	0	1	baris s_3
s_4	0	0	1	0	0	0	1	2	baris s_4

Iterasi simpleks dimulai dari *nonbasic variable* $(x_1, x_2) = (0, 0)$. *Nonbasic variable* adalah variabel-variabel yang harus dinolkan. Sedangkan variabel lainnya (s_1, s_2, s_3, s_4) disebut sebagai *basic variable*.

Tahap-tahap yang dilakukan selanjutnya dalam metode simpleks adalah sebagai berikut :

- Menentukan *entering variable*, yaitu *nonbasic variable* yang memiliki koefisien paling negatif pada *maximization objective equation* atau paling positif pada *minimization objective equation* (dilihat dari baris z).
- Menentukan *leaving variable*, yaitu dengan membandingkan semua sisi kanan persamaan *constraint* dengan *entering variable*, dan *basic variable* yang mempunyai nilai perbandingan (*ratio*) nonnegatif yang terkecil adalah *leaving variable.*

Misalnya, dalam contoh kasus Reddy Mikks ini *entering variable* untuk iterasi pertama adalah x_1, karena –5 adalah koefisien paling negatif. Sedangkan *leaving variable* adalah s_1, karena nilai perbandingan yang dihasilkan adalah $24/6 = 4$ yang merupakan nilai nonnegatif terkecil.

Untuk menghasilkan baris baru pada iterasi berikutnya, sehingga kolom *Basic* dan kolom *Solution* diperbarui, digunakanlah *Gauss Jordan row operation.* Untuk baris *pivot* (*pivot row*) digunakanlah persamaan 2.16.

$$\text{baris } pivot \text{ baru} = \text{baris } pivot \text{ sekarang} / pivot\ element \qquad (2.16)$$

sedangkan untuk baris lainnya (termasuk z) digunakanlah persamaan 2.17.

$$\text{baris baru} = \text{(baris sekarang)} - \text{(koefisien kolom } pivot\text{)} \times \text{(baris } pivot \text{ baru)} \qquad (2.17)$$

Pada kasus Reddy Mikks, baris *pivot* saat ini adalah baris s_1, sedangkan kolom *pivot* adalah kolom x_1, sehingga *pivot element* adalah 6. Berikut adalah tabel simpleks baru yang terbentuk :

Basic	z	x_1	x_2	s_1	s_2	s_3	s_4	Solution	
z	1	0	$-\frac{2}{3}$	$\frac{5}{6}$	0	0	0	20	baris z
x_1	0	6	$\frac{2}{3}$	$\frac{1}{6}$	0	0	0	4	baris x_1
s_2	0	1	$\frac{4}{3}$	$-\frac{1}{6}$	1	0	0	2	baris s_2
s_3	0	-1	$\frac{5}{3}$	$\frac{1}{6}$	0	1	0	5	baris s_3
s_4	0	0	1	0	0	0	1	2	baris s_4

Hal selanjutnya adalah menentukan *entering variable* dan *leaving variable* dari tabel simpleks tersebut. Kemudian dilanjutkan dengan memperbarui tabel simpleks. Begitu seterusnya hingga nilai koefisien pada baris z tidak ada yang negatif, sehingga hasil optimal telah tercapai.

2.5.3 Kasus Khusus dalam Metode Simpleks

Terdapat beberapa kasus khusus yang mungkin terjadi pada metode simpleks. Kasus-kasus tersebut adalah degenerasi, adanya solusi optimum alternatif, solusi tidak terbatas, dan solusi tidak fisibel.

2.5.3.1 Degenerasi

Kasus ini terjadi apabila satu atau lebih variabel basis (*basic variable*) berharga nol sehingga iterasi yang dilakukan selanjutnya bisa menjadi suatu *loop* yang akan kembali pada bentuk sebelumnya. Kejadian seperti ini disebut *cycling* atau *circling*.

Pada saat iterasi simpleks menghasilkan suatu solusi yang *degenerate* (salah satu variabel basis berharga nol), perhitungan tetap dilakukan, karena tidak semua persoalan menghasilkan solusi *degenerate* yang tetap (degenerasi temporer).

2.5.3.2 Solusi Optimum Alternatif

Kasus ini terjadi apabila fungsi tujuan paralel dengan fungsi pembatas, di mana paling sedikit salah satu dari variabel nonbasis (pada persamaan z pada iterasi terakhir) mempunyai koefisien berharga nol. Akibatnya walaupun variabel tersebut dinaikkan harganya (dijadikan variabel basis), harga z tetap tidak

berubah. Karena itu, solusi-solusi optimum yang lain ini biasanya dapat diidentifikasi dengan cara menunjukkan iterasi-iterasi tambahan pada metode simpleksnya, di mana variabel-variabel nonbasis yang berkoefisien nol itu selalu dipilih untuk menjadi *entering variable*.

2.5.3.3 Solusi Tidak Terbatas

Kasus init terjadi apabila ruang solusi tidak terbatas sehingga nilai fungsi tujuan dapat meningkat (untuk maksimasi) atau menurun (untuk minimasi) secara tidak terbatas. Apabila persoalannya dapat diselesaikan secara grafis (berdimensi dua), maka kasus ini akan mudah terdeteksi. Akan tetapi, jika persoalan yang dihadapi berdimensi tiga atau lebih, maka untuk mendeteksi apakah solusinya terbatas atau tidak, dilakukan dengan cara :

a. Memperhatikan koefisien-koefisien pembatas dari variabel nonbasis pada suatu iterasi. Jika koefisien-koefisien tersebut berharga negatif atau nol, berarti solusinya tak terbatas.
b. Jika koefisien fungsi tujuan variabel tersebut berharga negatif (untuk maksimasi) atau positif (untuk minimasi), maka nilai fungsi tujuannya tidak terbatas.

2.5.3.4 Solusi Infisibel

Model pemrograman linier dengan fungsi pembatas yang tidak konsisten tidak memiliki solusi fisibel. Kondisi ini tidak akan terjadi bila semua fungsi pembatasnya bertipe $\leq$ (dengan asumsi sisi kanan persamaan non negatif) karena *slack* akan memberikan solusi fisibel Untuk tipe pembatas yang lain dapat menggunakan variabel buatan.

Pada umumnya, ruang solusi infisibel terjadi karena kesalahan dalam memformulasikan model.

2.5.4 Solusi Dengan Variabel Buatan

Pemrograman linier yang memiliki pembatas (*constraint*) bertanda ($\leq$) dengan sisi kanan non negatif dapat diselesaikan dengan memberi variabel

slack. Namun untuk persoalan yang bertanda ≥ atau =, terdapat metode tertentu menggunakan variabel buatan. Dua metode yang akan digunakan yaitu metode M dan metode dua fase.

2.5.4.1 Metode M

Metode M diawali dengan bentuk persamaan biasanya. Persamaan ke-*i* yang tidak memiliki *slack* (atau variabel lain yang berperan sebagai *slack*) ditambah dengan variabel buatan R_i untuk solusi awalnya.

M adalah koefisien fungsi tujuan dari variabel buatan, dimana M adalah bilangan positif yang sangat besar (mendekati tak hingga). Nilai M negatif untuk persoalan maksimasi, dan positif untuk persoalan minimasi.

2.5.4.2 Metode Dua Fase

Karena terdapat kemungkinan kesalahan atau ketidakakuratan metode M, dimana manipulasi koefisien yang besar dan kecil dilakukan bersamaan, maka digunakanlah metode dua fase. Cara penyelesaian persoalan pemrograman linier akan dilakukan dalam dua fase, yaitu:

Fase 1 :

Fase ini digunakan untuk menguji apakah persoalan yang dihadapi memiliki solusi fisibel atau tidak. Pada fase ini fungsi tujuan semula diganti dengan meminimumkan jumlah variabel buatannya. Jika nilai minimum fungsi tujuan baru ini berharga nol (artinya seluruh variabel buatan berharga nol), berarti persoalan memiliki solusi fisibel, berlanjut ke fase 2. Namun, jika nilai minimum fungsi tujuan baru berharga positif, maka persoalan tidak memiliki solusi fisibel.

Fase 2 :

Solusi optimum dari fase 1 digunakan sebagai solusi awal persoalan. Dalam hal ini bentuk fungsi tujuan fase 1 dikembalikan pada fungsi tujuan semula. Kemudian pemecahan persoalan dilakukan seperti biasa.

BAB III

ANALISIS PERENCANAN PRODUKSI DI PT. IGLAS (PERSERO)

BAB III

ANALISIS PERENCANAAN PRODUKSI

DI PT. IGLAS (PERSERO)

Dalam bab ini dibahas tentang hal-hal yang berkaitan dengan aktifitas perencanaan produksi yang dihadapi PT. IGLAS (Persero). Pada bagian ini akan diuraikan tentang bagaimana pembentukan biaya-biaya lembur dan penyimpanan serta pembentukan stok pengaman. Selain itu juga dibahas mengenai identifikasi variabel keputusan dan parameter perencanaan produksi, serta pencarian model ARIMA untuk peramalan permintaan.

3.1 Sekilas Proses Produksi di PT. IGLAS (Persero)

PT. IGLAS merupakan perusahaan manufaktur yang bergerak di bidang pembuatan gelas kemas. Didirikan sejak 2 November 1955 dan memperoleh status Perusahaan Negara pada 1 Januari 1961, kemudian sejak 1979 berada di bawah pembinaan Departemen Perindustrian. Perusahaan ini memperoleh Sertifikat ISO-9002 : 1994 pada tahun 1995 serta Sertifikat Zero Accident selaku produsen gelas kemas sejak tahun 1997.

Berdasarkan strategi respons perusahaan terhadap permintaan konsumen, PT. IGLAS (Persero) tergolong dalam perusahaan yang memilih strategi *Make to Order*. Strategi respons perusahaan terhadap permintaan konsumen didefinisikan sebagai cara suatu perusahaan industri manufaktur dalam memberikan tanggapan atau respons terhadap permintaan konsumen. Perusahaan industri yang memilih strategi *Make to Order*, memiliki desain produk dan beberapa material standard dalam persediaan, dari produk-produk yang telah dibuat sebelumnya [GAS-02]. Aktifitas proses pembuatan produk bersifat khusus yang disesuaikan dengan setiap permintaan dari konsumen.

Secara garis besar aktifitas-aktifitas dalam manajemen permintaan dapat dikategorikan dalam dua aktifitas utama, yaitu pelayanan pesanan dan peramalan. Aktifitas peramalan dilakukan pada permintaan dari item-item yang merupakan *independent demand*, yaitu permintaan yang tidak terkait langsung dengan permintaan untuk item lain.

Sebelum membahas mengenai analisa perencanaan produksi di PT. IGLAS (Persero), terlebih dahulu dikemukakan mengenai garis besar proses produksi yang berjalan di PT. IGLAS (Persero).

Dalam proses produksinya, PT. IGLAS (Persero) menggunakan bahan baku utama beling, berupa *silica sand* (pasir kuarsa) dan *cullet* (pecahan gelas), batu kapur, dan soda abu. Bahan-bahan tersebut dimasukkan ke dalam mesin penghancur, kemudian diolah untuk diteruskan ke dalam proses peleburan. Bahan yang telah lebur, kemudian dibentuk menjadi potongan-potongan gelas yang sesuai berat gelas kemas, dan segera dikirim ke tempat pembentukan gelas kemas untuk dicetak dalam mesin-mesin pembentuk yang telah disiapkan dengan cetakan. Gelas kemas yang telah terbentuk kemudian didinginkan secara perlahan pada sebuah ban berjalan. Pemeriksaan yang teliti dilakukan atas setiap gelas kemas yang telah melewati proses pendinginan. Dan setiap gelas kemas yang ditolak akan didaur ulang dan dimasukkan lagi ke dalam sistem proses saat itu juga. Gelas kemas yang telah lolos pemeriksaan, beberapa diantaranya akan diberi label berwarna, dan selanjutnya dikemas dan ditampung, untuk kemudian dikirimkan.

3.2 Pembentukan Biaya Lembur dan Biaya Penyimpanan Persediaan Pada Proses Produksi

Dalam proses pembuatan produk yang dilakukan PT. IGLAS (Persero), terdapat beberapa biaya yang harus dialokasikan antara lain adalah biaya bahan baku, biaya mesin, dan biaya tenaga kerja. Selain itu, biaya penyimpanan persediaan hasil produksi juga diperhitungkan.

Biaya bahan baku meliputi biaya-biaya yang dikeluarkan untuk mendapatkan bahan baku utama (*silica sand*, *cullet*, batu kapur, dan soda abu) dan

bahan baku tambahan. Biaya mesin meliputi biaya-biaya yang dikeluarkan untuk operasional mesin-mesin produksi, termasuk diantaranya biaya listrik dan biaya bahan bakar mesin. Sedangkan biaya tenaga kerja diantaranya meliputi gaji pegawai / tenaga kerja, uang tunjangan kesejahteraan dan kesehatan pegawai, maupun lembur.

Biaya penyimpanan persediaan terjadi ketika di dalam gudang / tempat penyimpanan terdapat bahan atau barang yang disimpan. Untuk setiap satuan hasil produksi yang disimpan di dalam gudang membutuhkan biaya penyimpanan untuk pemeliharaan. Semakin banyak hasil produksi yang disimpan di dalam gudang maka biaya penyimpanan persediaan juga semakin besar.

Permintaan yang masuk pada divisi penjualan, selanjutnya akan dikerjakan oleh divisi produksi. Apabila jumlah permintaan yang masuk sangat besar dimungkinkan kebutuhan waktu produksi melebihi waktu kerja reguler, sehingga terdapat penambahan waktu kerja pegawai (lembur).

Jumlah permintaan sedapat mungkin dapat dipenuhi oleh divisi produksi dengan meminimumkan biaya lembur pegawai. Biaya lembur akan tergantung pada jumlah yang harus diproduksi. Sedangkan jumlah yang harus diproduksi tergantung pada peramalan dan jumlah persediaan yang ada di gudang.

Semakin banyak jumlah persediaan di gudang, maka permintaan (hasil peramalan) semakin mudah untuk dipenuhi, karena jumlah yang harus diproduksi semakin sedikit atau bahkan tanpa produksi sama sekali. Namun, seperti pernyataan sebelumnya, jumlah persediaan yang semakin banyak akan memperbesar jumlah biaya penyimpanan persediaan.

3.3 Pembentukan Stok Pengaman

Pertimbangan bahwa jumlah permintaan aktual akan berbeda dengan jumlah permintaan hasil peramalan sangatlah diperlukan. Selisih antara jumlah permintaan aktual dengan jumlah permintaan hasil peramalan, lebih lanjut akan mempengaruhi timbulnya stok berlebih (*over stock*) maupun kekurangan stok (*under stock*). Dalam kasus *under stock*, diperlukan sejumlah stok yang diharapkan dapat memenuhi jumlah permintaan yang tidak terduga, yang biasanya

disebut stok pengaman (*safety stock*). Jumlah stok pengaman akan didasarkan pada selisih dari jumlah permintaan hasil peramalan dengan jumlah permintaan aktual tadi.

Jumlah stok pengaman akan menambah jumlah persediaan, yang tentunya akan menambah biaya penyimpanan persediaan. Sehingga diperlukan kebijaksanaan yang dapat meminimumkan jumlah stok pengaman. Persamaan (3.1) merupakan formulasi untuk mendapatkan tingkat kebutuhan stok pengaman untuk produk *j* pada periode *t*.[PIS-01]

$$SS_t = sf * \sigma_j \tag{3.1}$$

dimana :

SS_t = Tingkat kebutuhan stok pengaman untuk periode *t*

sf = Faktor pengaman

σ_j = Standard deviasi dari kesalahan peramalan untuk produk *j*

Dalam kasus tertentu sangatlah dimungkinkan, apabila jumlah permintaan melebihi jumlah persediaan, dan stok pengaman juga tidak dapat mencukupi jumlah permintaan tersebut. Biaya yang timbul akibat kondisi seperti ini disebut dengan *shortage cost*. Biaya tersebut sebenarnya merupakan *penalty* yang harus ditanggung perusahaan karena tidak dapat memenuhi permintaan. Perusahaan kehilangan kesempatan memperoleh pendapatan (atau bahkan harus mengeluarkan biaya lebih), serta adanya resiko kehilangan kepercayaan pelanggan.

Berdasarkan uraian-uraian di atas mengenai garis besar proses produksi, pembentukan biaya lembur dan biaya penyimpanan persediaan, serta pembentukan stok pengaman, selanjutnya dicari bagaimana model matematis yang tepat untuk merepresentasikan kondisi dan batasan-batasan dalam meminimumkan jumlah total biaya-biaya tersebut, dalam hal ini biaya lembur dan biaya penyimpanan persediaan.

3.4 Model Perencanaan Produksi

Permasalahan perencanaan produksi, yang dalam hal ini dikhususkan pada penentuan total biaya minimum untuk tiga komponen biaya yaitu biaya lembur dan penyimpanan persediaan, diformulasikan sebagai model pemrograman linier. Pemrograman linier digunakan untuk mendapatkan hasil yang optimum pada fungsi tujuannya, dengan penetapan nilai variabel-variabel keputusan tertentu dan memperhatikan pembatas-pembatasnya.

Model perencanaan produksi ini diawali dengan mendefinisikan variabel-variabel keputusan dan parameter-parameter yang mempengaruhi fungsi tujuan.

3.4.1 Variabel-variabel Keputusan

Variabel-variabel keputusan untuk model perencanaan produksi tampak pada tabel 3.1.

Tabel 3.1 Variabel Keputusan Untuk Perencanaan Produksi

Variabel Keputusan	Deskripsi
$X_{i,t}$	Jumlah produk *i* yang harus diproduksi pada periode *t* (unit)
$I_{i,t}$	Jumlah persediaan produk *i* pada saat ke *t* (unit)
O_t	Waktu kerja lembur selama periode *t* (*man-hours*)

Jumlah persediaan produk, $I_{i,t}$, merupakan jumlah produk *i* yang masih tersisa dan tersimpan dalam gudang pada saat periode *t*. Waktu lembur, O_t digunakan jika pada periode *t* terdapat kebutuhan tenaga pegawai di luar waktu kerja rutin pegawai.

3.4.2 Parameter-parameter

Untuk menyusun model perencanaan dan mendapatkan nilai minimum untuk total tiga komponen biaya tadi, perlu ditentukan parameter apa saja yang mempengaruhi nilainya. Parameter-parameter yang digunakan dalam model perencanaan produksi ini didaftarkan pada tabel 3.2.

Tabel 3.2 Parameter Perencanaan Produksi

Parameter	Deskripsi
h_i	Besar biaya penyimpanan persediaan produk *i* untuk tiap unit dan tiap periode (rupiah/unit/periode).
c_o	Besar biaya lembur pegawai (rupiah/*man-hour*)
$d_{i,t}$	Hasil peramalan permintaan produk *i* pada periode *t* (unit)
a_i	Waktu yang dibutuhkan untuk proses produksi produk *i* (*hours*/unit)
R_t	Total waktu kerja reguler yang tersedia pada periode *t* (*man-hours*)
$(om)_t$	Waktu lembur maksimum yang tersedia pada periode *t* (*man-hours*)
$SS_{i,t}$	Jumlah stok pengaman untuk produk *I* pada periode *t* (unit)
$I_{i,0}$	Jumlah persediaan awal produk *i* (unit)
T	Total jumlah periode dalam perencanaan
N	Total jumlah produk dalam perencanaan

Nilai dari parameter-parameter pada tabel 3.2 di atas ditentukan berdasarkan kebijakan PT. IGLAS (Persero) dan diluar kejadian-kejadian luar biasa (bencana alam, kebakaran, kenaikan harga bahan bakar, dan lain-lain).

Dipilihnya parameter-parameter pada tabel 3.2 di atas berdasarkan hasil eksplorasi data serta dengan alasan-alasan tertentu.

a. Biaya penyimpanan persediaan (h_i)

Seperti yang telah dikemukakan pada subbab 3.1.2, bahwa ketika persediaan disimpan dalam gudang terjadi biaya penyimpanan. Biaya ini digunakan untuk memelihara persediaan di gudang. Besar biaya penyimpanan untuk produk *i* setiap satuan unit persediaan dan setiap satuan waktu (periode) ditetapkan sebesar h_i.

b. Biaya lembur (c_o)

Biaya lembur terjadi apabila jumlah waktu proses produksi yang dibutuhkan ternyata melebihi waktu kerja reguler. Besar biaya lembur untuk setiap satuan penggunaan tenaga kerja (*man-hours*) ditetapkan sebesar c_o rupiah.

c. Hasil peramalan permintaan ($d_{i,t}$)

Parameter ini merupakan masukan yang berasal dari hasil proses peramalan permintaan produk *i* untuk periode *t*. Metode peramalan berbasis ARIMA digunakan dalam manajemen permintaan..

d. Waktu proses produksi (a_i)

Untuk pembuatan setiap unit produk *i* dibutuhkan waktu sebanyak *a* jam. Dan waktu ini berpengaruh terhadap pembentukan biaya lembur.

e. Total waktu kerja reguler yang tersedia pada periode *t* (R_t)

Waktu kerja reguler ini merupakan waktu kerja pegawai yang telah ditentukan oleh perusahaan.

f. Waktu lembur maksimum yang tersedia pada periode *t* ($(om)_t$)

Penggunaan waktu lembur tidak boleh melebihi waktu kerja reguler yang tersedia.

g. Jumlah stok pengaman ($SS_{i,t}$)

Stok pengaman digunakan untuk mengatasi kemungkinan terjadi kekurangan stok (*under stock*) pada persediaan. Jumlah stok pengaman dihitung berdasarkan kesalahan peramalan.

h. Jumlah persediaan awal ($I_{i,0}$)

Sebagai inisialisasi perlu diketahui jumlah persediaan awal yang ada di gudang.

i. Total periode perencanaan (T)

Periode perencanaan dimaksudkan untuk memberi batasan sampai periode berapakah perencanaan produksi akan dilakukan.

j. Total jumlah jenis produk dalam perencanaan (N)

Jumlah jenis produk dimaksudkan untuk memberi batasan produk apa saja yang akan dimasukkan dalam perencanaan produksi.

3.4.3 Model Pemrograman Linier

Setelah variabel-variabel keputusan dan parameter didefinisikan, selanjutnya adalah menentukan model pemrograman linier untuk perencanaan produksinya.

Fungsi tujuan dari model ini adalah meminimasi total biaya penyimpanan dan biaya lembur dari semua produk dan periode. Persamaan 3.2 merupakan fungsi tujuan dari model pemrograman linier yang diajukan.

$$\textit{Minimasi } Z = \sum_{i=1}^{N}\sum_{t=1}^{T} h_i I_{i,t} + \sum_{t=1}^{T} c_o O_t \tag{3.2}$$

Sedangkan persamaan 3.3 sampai dengan persamaan 3.6 adalah pembatas-pembatas yang harus dipenuhi dalam perhitungan nilai optimalnya.

$$I_{i,t-1} + X_{i,t} - I_{i,t} = d_{i,t} \tag{3.3}$$

Persamaan 3.3 merupakan pembatas yang memberikan aturan hubungan antara jumlah persediaan produk i pada periode sebelum t ($I_{i,t-1}$), jumlah persediaan produk i pada periode t ($I_{i,t}$), jumlah produksi produk i pada perode t ($X_{i,t}$), dan hasil peramalan permintaan produk i pada periode t ($d_{i,t}$).

$$\sum_{i=1}^{N} a_i X_{i,t} - O_t \le R_t \tag{3.4}$$

Persamaan 3.4 merupakan pembatas yang memberikan aturan hubungan antara jumlah waktu produksi (hasil perkalian antara produktivitas, a_i, dengan

jumlah yang harus diproduksi, $X_{i,t}$) untuk semua jenis produk (sebanyak N jenis produk), sebelum memperhitungkan waktu lembur (O_t), tidak boleh melebihi waktu kerja reguler (R_t) untuk masing-masing periode t.

$$I_{i,t} \geq SS_{i,t} \tag{3.5}$$

Persamaan 3.5 merupakan pembatas yang memberikan aturan hubungan, dimana jumlah persediaan produk i pada periode t ($I_{i,t}$) adalah minimum sebanyak jumlah stok pengaman untuk produk tersebut dan periode itu ($SS_{i,t}$).

$$O_t \leq (om)_t \tag{3.6}$$

Persamaan 3.6 merupakan pembatas yang memberikan aturan waktu lembur maksimum pada periode t (om_t).

Selain itu, terdapat pembatas untuk setiap variabel keputusan tidak dapat bernilai negatif (*nonnegativity constraint*).

3.5 Pencarian Model ARIMA untuk Peramalan Permintaan

Hasil peramalan permintaan merupakan salah satu parameter dari pembentuk total biaya, dalam hal ini biaya lembur dan biaya penyimpanan persediaan. Dan parameter ini cenderung berubah untuk tiap periodenya, sehingga jumlah biaya yang dikeluarkan banyak bergantung pada hasil peramalan.

Box-Jenkins mengembangkan metode ARIMA untuk teknik peramalan. Metode ini yang digunakan untuk melakukan proses peramalan permintaan. Data yang digunakan untuk melakukan proses peramalan adalah data permintaan pembuatan botol atau gelas X pada periode waktu tertentu. Maksud dari pengambilan data didasarkan pada periode waktu, adalah karena pada proses peramalan lebih relevan digunakan data runtut waktu. ARIMA memiliki beberapa tahap yang diharapkan dapat menghasilkan model yang sesuai. Sehingga model ini dapat digunakan untuk meramalkan nilai pada periode berikutnya.

Sebagai contoh diberikan dua buah sampel data [LER-02] yang ditunjukkan oleh tabel 3.3 dan tabel 3.4. Jumlah record untuk sampel data yang

ditunjukkan oleh tabel 3.3 adalah 100 record, sedangkan pada tabel 3.4 jumlahnya adalah 65 record.

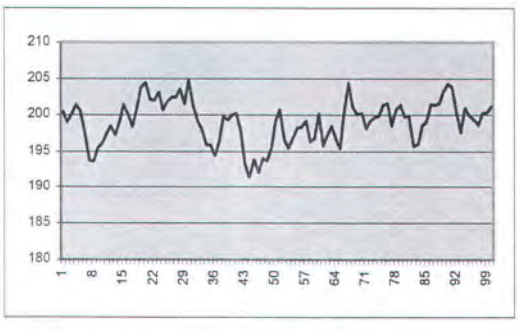
Tabel 3.3 Sampel Data Pertama dengan 100 record

200.32	197.44	202.05	201.3	200.1	200.62	195.75	198.07	199.85	203.8
198.96	198.42	201.89	199.07	197.75	196.62	197.19	199.1	195.69	200.17
200.05	197.27	203.05	197.93	193.19	195.49	198.39	199.6	196.02	197.6
201.31	198.93	200.64	195.97	191.34	196.75	196.71	199.81	198.46	200.88
200.61	201.34	201.68	195.86	193.81	198.09	195.41	201.28	199.03	199.85
197.62	199.98	202.34	194.46	192.03	198.29	200.81	201.55	201.37	199.24
193.7	198.31	202.36	196.3	193.96	199.11	204.38	198.46	201.32	198.57
193.62	200.59	203.49	199.65	193.68	196.4	201.1	200.68	201.57	200.2
195.5	203.77	201.5	199.25	195.48	196.76	200.04	201.34	203.26	200.3
196.17	204.41	204.8	199.93	199.11	200.07	200.1	199.65	204.27	201.08

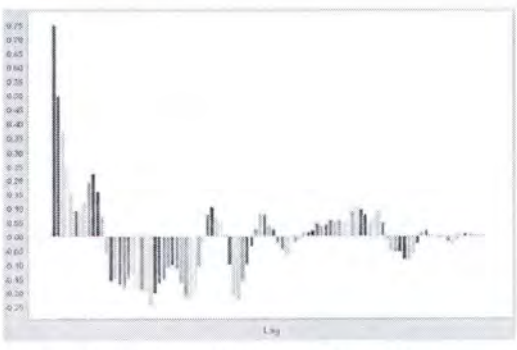
Tabel 3.4 Sampel Data Kedua dengan 65 record

235	320	355	220	255	285	250	290	240	310	190	245	340
320	275	175	400	285	250	355	225	275	220	295	170	190
115	205	285	275	250	225	280	270	225	320	275	175	250
355	295	200	185	300	125	370	180	285	215	205	270	300
190	240	290	370	225	295	250	270	250	260	265	225	195

Tahap awal adalah pemeriksaan stasioneritas data. Tahap ini diperlukan karena data yang dianalisa pada metode peramalan ini adalah data runtut waktu yang stasioner. Untuk mengetahui apakah data runtut waktu yang digunakan telah stasioner atau belum, dapat dilakukan dengan melihat plot data dan plot fungsi autokorelasi data (*correlogram*). Plot data dan *correlogram* ACF untuk sampel data pertama ditunjukkan oleh gambar 3.1 dan gambar 3.2.



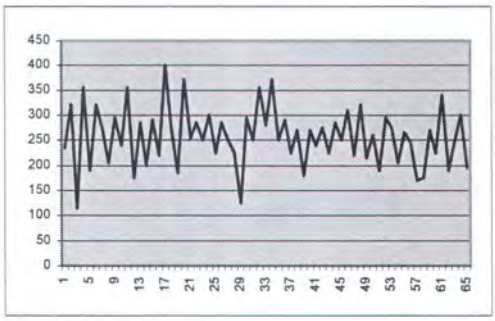
Gambar 3.1 Plot Sampel Data Pertama



Gambar 3.2 ACF Sampel Data Pertama

Plot data untuk sampel data pertama tidak menunjukkan sifat stasioner secara jelas, sedangkan *correlogram* ACF yang terbentuk menunjukkan

penurunan autokorelasi secara cepat. Penurunan autokorelasi yang cepat menunjukkan bahwa data tersebut bersifat stasioner. Sedangkan untuk sampel data kedua, plot data dan *correlogram* ACF ditunjukkan oleh gambar 3.3 dan 3.4.



Gambar 3.3 Plot Sampel Data Kedua

Gambar 3.4 ACF Sampel Data Kedua

Ternyata plot data dan *correlogram* ACF sampel data kedua juga menunjukkan hal yang sama, sehingga dapat disimpulkan jika sampel data kedua bersifat stasioner.

Seperti yang telah dinyatakan pada bab 2, data yang bersifat tidak stasioner memiliki autokorelasi yang signifikan dari nol pada lag awal, kemudian mengecil secara perlahan menuju ke nol. Apabila ternyata data belum stasioner maka perlu dilakukan proses *differencing* hingga tingkat tertentu. Contoh data yang tidak stasioner dan proses *differencing* akan diberikan di akhir sub bab ini.

Setelah diyakinkan bahwa data bersifat stasioner, tahap selanjutnya adalah identifikasi model. Identifikasi model dilakukan dengan melihat *correlogram* ACF dan PACF data dan membandingkannya dengan *correlogram* ACF dan PACF yang umum, yang telah ditunjukkan oleh gambar 2.2 pada bab 2.

Correlogram ACF untuk kedua sampel data telah ditunjukkan oleh gambar 3.2 dan gambar 3.4. Sedangkan *correlogram* PACF untuk kedua sampel data, masing-masing ditunjukkan oleh gambar 3.5 dan gambar 3.6.

Gambar 3.5 PACF Sampel Data Pertama

Gambar 3.6 PACF Sampel Data Kedua

Untuk sampel data pertama, ACF cenderung menghilang menuju ke nol secara sinusoidal dan PACF terpotong setelah lag pertama. Sehingga, jika membandingkannya dengan gambar 2.2, maka identifikasi untuk sampel data ini adalah model AR(1) atau ARIMA(1,0,0). Sedangkan untuk sampel data kedua, jika membandingkan ACF dan PACF yang diperoleh dengan gambar 2.2, maka identifikasi modelnya adalah MA(2) atau ARIMA(0,0,2).

Tahap selanjutnya adalah estimasi parameter. Pada sampel data pertama, data diidentifikasi sebagai model AR(1), karenanya estimasi parameter dilakukan untuk menemukan nilai konstanta ϕ_0 dan koefisien ϕ_1. Estimasi parameter untuk AR(1) telah disampaikan pada sub bab 2.4.3.1 pada bab 2. Hasil perhitungan yang didapatkan untuk koefisien ϕ_1 adalah 0,7522 , sedangkan hasil perhitungan yang didapatkan untuk konstanta ϕ_0 adalah 49,3128.

Kemudian, pada model AR(1) dengan nilai konstanta dan koefisien yang telah ditemukan tersebut, residual yang didapatkan diuji dengan *Q-Statistic*. Uji *Q-Statistic* telah diberikan pada sub bab 2.4.4. Nilai *Q-Statistic* untuk autokorelasi residual 30 lag pertama adalah sebesar 27,8013, sedangkan nilai *chi-square* pada *siqnificant level* 0,05 dan derajat kebebasan 30 yang diperoleh dari tabel adalah 43,773. Karena nilai *Q-Statistic* lebih kecil dari nilai *chi-square* yang diperoleh dari tabel, maka semua koefisien autokorelasi residual yang didapat dianggap tidak berbeda dengan nol, atau dapat dikatakan residualnya berpola acak, sehingga model telah dispesifikasikan dengan benar.

Pada sampel data kedua, data diidentifikasi sebagai model MA(2), karenanya estimasi parameter dilakukan untuk menemukan nilai μ, ω_1, dan ω_2. Sub bab 2.4.3.4 menjelaskan estimasi parameter untuk model MA(2). Dan akhirnya hasil yang didapatkan [LER-02] adalah konstanta μ = 257,380, koefisien ω_1 = 0,290, dan koefisien ω_2 = –0,302.

Kemudian, pada model MA(2) dengan nilai konstanta dan koefisien yang telah ditemukan tersebut, residual yang didapatkan diuji dengan *Q-Statistic*. Nilai *Q-Statistic* untuk autokorelasi residual 30 lag pertama adalah sebesar 16,2856, sedangkan nilai *chi-square* pada *siqnificant level* 0,05 dan derajat kebebasan 30 yang diperoleh dari tabel adalah 43,7729. Karena nilai *Q-Statistic* lebih kecil dari nilai *chi-square* yang diperoleh dari tabel, maka semua koefisien autokorelasi residual yang didapat dianggap tidak berbeda dengan nol, atau dapat dikatakan residualnya berpola acak, sehingga model telah dispesifikasikan dengan benar.

Setelah kedua sampel data ditemukan model yang sesuai, maka selanjutnya model masing-masing sampel data tersebut dapat digunakan untuk melakukan peramalan, yaitu meramalkan data yang ke-101 untuk sampel data pertama dan data yang ke-66 untuk sampel data kedua.

Berdasarkan estimasi parameter model AR(1) yang telah dilakukan sebelumnya, persamaan model untuk sampel data pertama dapat dituliskan oleh persamaan 3.7.

$$Y_t = 49{,}3128 + 0{,}7522Y_{t-1} \tag{3.7}$$

Dengan menggunakan persamaan 3.7, hasil peramalan untuk data yang ke-101, dihitung sebagai berikut :

$$Y_{101} = 49{,}3128 + 0{,}7522Y_{100}$$
$$Y_{101} = 49{,}3128 + 0{,}7522 \times 201{,}08$$
$$Y_{101} = 200{,}56518$$

Sedangkan persamaan model untuk sampel data kedua berdasarkan estimasi parameter model MA(2) yang telah dilakukan sebelumnya, dituliskan oleh persamaan model 3.8.

$$Y_t = 257{,}380 - 0{,}290e_{t-1} + 0{,}302e_{t-2} \tag{3.8}$$

Dengan menggunakan persamaan model 3.8 tersebut, perhitungan hasil peramalan untuk data ke-66, dengan e_{-1} dan e_0 dianggap 0, dituliskan sebagai berikut :

$$Y_{66} = 257{,}380 - 0{,}290e_{65} + 0{,}302e_{64}$$
$$Y_{66} = 257{,}380 - 0{,}290 \times (-36{,}401) + 0{,}302 \times 43{,}059$$
$$Y_{66} = 280{,}940$$

Peramalan untuk kedua sampel data ini, menggunakan model ARIMA yang tidak memiliki komponen I (hanya komponen AR dan MA saja). Komponen I disertakan bila data aslinya tidak stasioner, sehingga datanya harus diubah menjadi stasioner melalui proses *differencing*. Contoh data yang tidak stasioner [NGU-02] dan proses *differencing* ditunjukkan oleh tabel 3.5.

Tabel 3.5 Contoh Data dan Proses Differencing

Periode	Y_t	$Y_t - Y_{t-1}$	$(Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2})$
1	8		
2	12	12 – 8 = 4	
3	15	15 – 12 = 3	3 – 4 = -1
4	19	19 – 15 = 4	4 – 3 = 1
5	25	25 – 19 = 6	6 – 4 = 2
6	30	30 – 25 = 5	5 – 6 = -1
7	34	34 – 30 = 4	4 – 5 = -1
8	40	40 – 34 = 6	6 – 4 = 2
9	45	45 – 40 = 5	5 – 6 = -1
10	51	51 – 45 = 6	6 – 5 = 1

Kolom 2 menunjukkan data awal sebelum mengalami proses *differencing*. Data tersebut menunjukkan adanya trend yang sangat jelas, nilai data semakin meningkat seiring bertambahnya periode. Sedangkan kolom 3 menunjukkan data hasil *differencing* tingkat pertama. Walaupun tidak cukup jelas namun pola trend masih terlihat di sini, sehingga dilakukan *differencing* tingkat kedua yang ditunjukkan oleh kolom 4.

BAB IV

ANALISIS DAN DESAIN APLIKASI

BAB IV

ANALISIS DAN DESAIN APLIKASI

Bab ini membahas tentang analisis dan desain aplikasi yang ditujukan sebagai perangkat lunak yang merepresentasikan penerapan metode ARIMA dalam meramalkan jumlah permintaan yang berperan sebagai masukan dalam perencanaan produksi sehingga hasil yang dicapai dapat optimal. Pembahasan diawali dengan analisis terhadap proses-proses yang akan dilakukan oleh aplikasi, termasuk spesifikasi masukan dan luaran. Dilanjutkan dengan pembuatan desain aplikasi yang merupakan hasil dari analisis yang telah dilakukan sebelumnya, termasuk desain antarmuka dari aplikasi yang akan dibuat.

Dalam analisis dan desain untuk aplikasi, bahasa pemodelan yang digunakan adalah UML (*Unified Modelling Language*). UML adalah bahasa standard untuk visualisasi, spesifikasi, perancangan, dan dokumentasi dari kerangka sistem perangkat lunak [QUA-02]. Salah satu perangkat lunak yang digunakan untuk analisis dan desain proses yang menggunakan bahasa pemodelan UML adalah Rational Rose Enterprise Edition 2002. UML digunakan untuk memberikan kerangka sistem yang cukup spesifik, sehingga mendekati implementasi pengembangan aplikasi untuk sistem tersebut.

4.1 Analisis Aplikasi

Aplikasi yang dibuat adalah representasi dari simulasi penerapan metode ARIMA dalam meramalkan jumlah permintaan sebagai salah satu parameter yang berperan dalam mengoptimalkan biaya lembur dan biaya penyimpanan persediaan. Proses peramalan membutuhkan suatu model yang dapat mewakili pola dari data yang ditelilti. Dalam membuat model, diperlukan sejumlah data yang telah lalu. Sehingga dari data yang telah lalu tersebut pola dari data dapat diketahui. Hasil dari peramalan adalah suatu nilai atau kondisi yang disesuaikan dengan model yang ditemukan. Kemudian hasil peramalan tersebut dapat diuji

dengan data sesungguhnya untuk melihat keakuratan hasil ramalan. Data yang digunakan untuk menemukan pola atau model selanjutnya disebut data pelatihan, sedangkan data yang digunakan untuk menguji adalah data pengujian.

Data permintaan digunakan sebagai data pelatihan dan data pengujian. Data pelatihan dimaksudkan untuk membentuk model peramalan yang paling sesuai dengan pola data aktual, sedangkan data pengujian digunakan untuk menguji model yang telah dibentuk oleh data pelatihan tadi. Selain itu, hasil dari pengujian juga memberikan nilai selisih antara data hasil peramalan dengan data aktual.

Berikutnya adalah data parameter perencanaan produksi, yang digunakan untuk menghitung total biaya lembur dan biaya penyimpanan. Parameter-parameter tersebut telah didaftarkan pada tabel 3.2 pada sub bab 3.4.2. Dua buah parameter yaitu hasil peramalan (d_t) dan jumlah stok pengaman (SS_t) didapatkan dari proses peramalan.

Di dalam proses peramalan menggunakan metode ARIMA, akan didapatkan suatu persamaan model. Model ini tersusun dari beberapa nilai-nilai parameter yang banyaknya tergantung dari model ARIMA yang digunakan. Persamaan model ARIMA ini memiliki bentuk umum tertentu, dan hanya berbeda pada nilai tiap parameternya. Sehingga persamaan model dapat disimpan dengan cara mencatat nilai-nilai dari parameter modelnya. Sedangkan mengenai langkah-langkah dalam metode ARIMA Box-Jenkins telah diuraikan pada bab sebelumnya.

Penyimpanan data runtut waktu maupun parameter-parameter lain dilakukan dalam database, dan database yang digunakan adalah MySQL. MySQL merupakan database yang cukup ringan dan *portable*. Karena data yang digunakan tidak terlalu banyak, sehingga penggunaan MySQL sebagai database dianggap cukup sesuai.

Secara garis besar, terdapat beberapa proses yang akan dilakukan oleh aplikasi, proses-proses tersebut adalah :

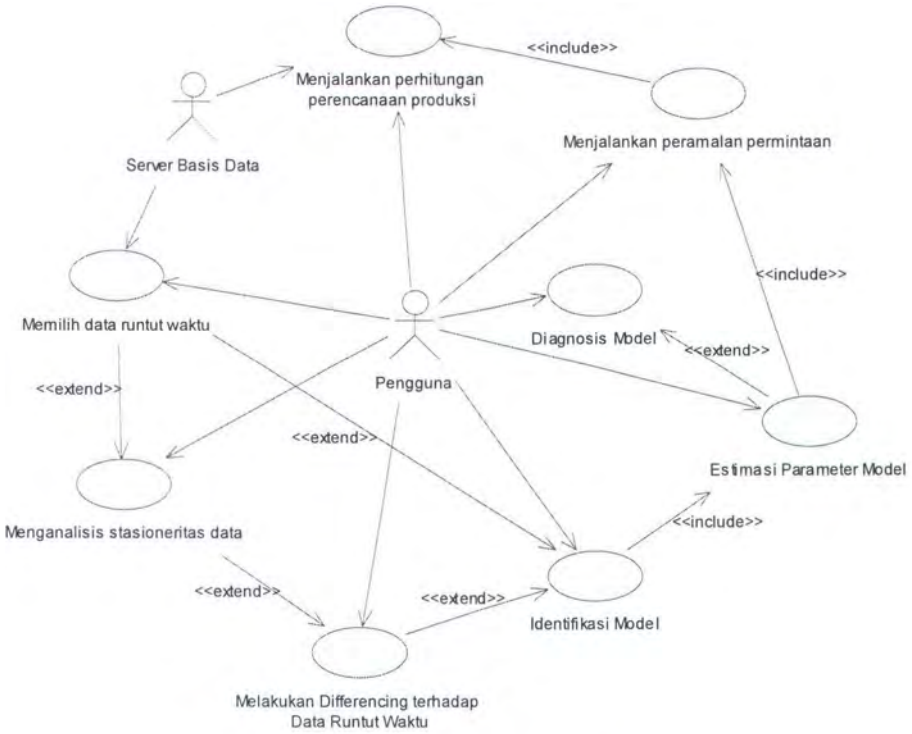
a. Memilih data runtut waktu
b. Menganalisis stasioneritas data

c. Melakukan *differencing* terhadap data runtut waktu
d. Identifikasi model
e. Estimasi parameter model
f. Diagnosis model
g. Menjalankan persamaan permintaan
h. Menjalankan perhitungan perencanaan produksi

Analisis yang digunakan terhadap proses-proses tersebut melibatkan beberapa *use case* dan *actor*. *Use case* adalah sebuah pola yang menggambarkan perilaku sistem. *Actor* adalah seseorang atau sesuatu yang harus berinteraksi dengan sistem yang sedang dikembangkan. Setiap *use case* adalah sehimpunan transaksi yang dilakukan oleh *actor* dan sistem. Diagram *use case* dibuat untuk menggambarkan hubungan antara beberapa *actor* dan beberapa *use case*. Gambar 4.1 merupakan diagram *use case* aplikasi untuk proses-proses tersebut.

Pada diagram *use case* pada gambar 4.1 itu, terdapat sembilan buah *use case* dan dua buah *actor*. Di antara asosiasi yang menghubungkan antar *use case* tersebut terdapat *stereotype include* dan *extend*. Hubungan *include* menggambarkan bahwa suatu *use case* secara keseluruhan meliputi fungsionalitas dari *use case* lainnya. Sedangkan hubungan *extend* antar *use case* berarti bahwa satu *use case* merupakan tambahan fungsionalitas dari *use case* yang lain jika suatu kondisi atau syarat tertentu terpenuhi.

Dari gambar 4.1 tampak bahwa hubungan *include* dimiliki oleh asosiasi antara *use case Identifikasi model* dengan *use case Estimasi parameter model*, antara *use case Estimasi parameter model* dengan *use case Menjalankan peramalan permintaan*, dan antara *use case Menjalankan peramalan permintaan* dengan *use case Menjalankan perhitungan perencanaan produksi*. Ketiga hubungan *include* tersebut menunjukkan bahwa *use case* yang bersangkutan saling berurutan. Sedangkan asosiasi yang menghubungkan antar uses lainnya memiliki *stereotype extend*, sehingga *use case* yang bersangkutan merupakan tambahan dari alur proses.



Gambar 4.1 Diagram Use Case Aplikasi

Untuk memperjelas mengenai *use case-use case* yang terdaftar diberikan beberapa uraian mengenai spesifikasi dan diagram aktivitas untuk setiap *use case* tersebut.

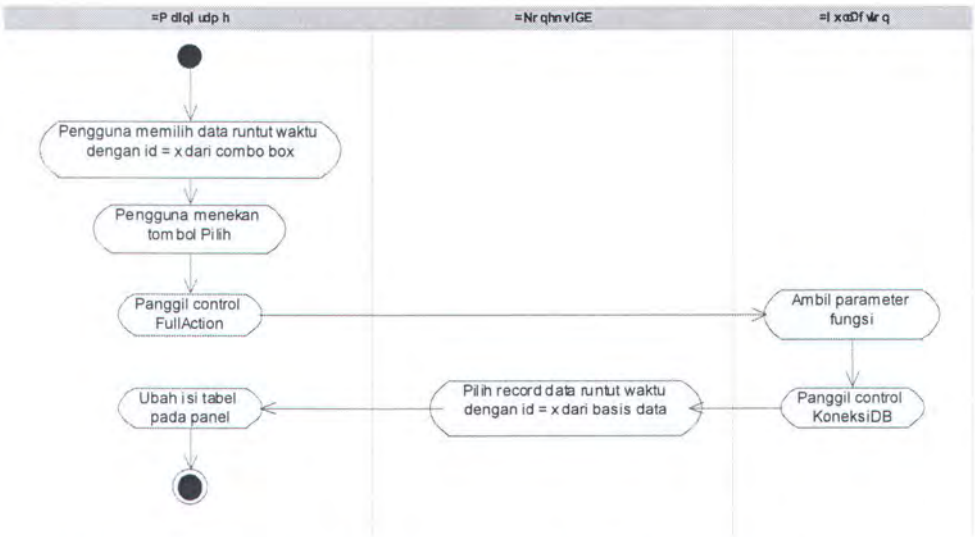
4.1.1 Use Case Memilih Data Runtut Waktu

Use case ini mewakili proses pemilihan data runtut waktu yang akan digunakan. Spesifikasi dari use case ini ditunjukkan oleh tabel 4.1.

Tabel 4.1 Spesifikasi Use Case Memilih Data Runtut Waktu

Spesifikasi	Penjelasan
Aktor	• Pengguna • Basis Data
Tujuan	Memilih data runtut waktu yang akan diramalkan untuk periode berikutnya
Ringkasan	*Use case* ini merupakan proses pemilihan data runtut waktu yang tersimpan dalam basis data untuk digunakan dalam proses peramalan
Masukan	Id data runtut waktu
Luaran	Record data runtut waktu
Kondisi Awal	• Isi tabel pada panel data runtut waktu berupa record data runtut waktu *default* (id = 1) • Isi tabel pada panel data runtut waktu berupa record data runtut waktu pilihan pengguna yang terakhir
Kondisi Akhir	Isi tabel pada panel data runtut waktu berupa record data runtut waktu sesuai pilihan pengguna
Aliran Aksi Normal	• Pengguna memilih id dan nama data runtut waktu dari combo box • Pengguna menekan tombol "Pilih" • Isi tabel pada panel data runtut waktu berubah

Aktivitas yang dilakukan dalam *use case Memilih data runtut waktu* ditunjukkan oleh gambar 4.2.



Gambar 4.2 Diagram Aktivitas Use Case Memilih Data Runtut Waktu

4.1.2 Use Case Menganalisis Stasioneritas Data

Use case ini merupakan proses analisis stasioneritas terhadap data runtut waktu. Spesifikasi *use case* ini ditunjukkan oleh tabel 4.2.

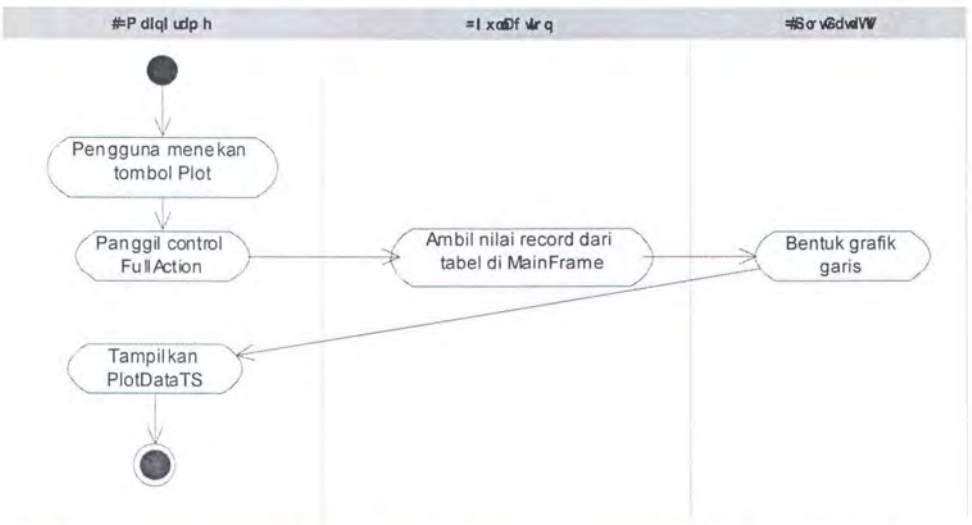
Tabel 4.2 Spesifikasi Use Case Menganalisis Stasioneritas Data

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Mengetahui stasioneritas data runtut waktu yang terpilih
Ringkasan	Use case ini merupakan proses analisis stasioneritas terhadap data runtut waktu terpilih yang memberikan gambaran berupa plot data atau berupa *correlogram.*
Masukan	Nilai-nilai record data runtut waktu terpilih
Luaran	• Plot Data • Correlogram
Kondisi Awal	Pengguna telah memilih data runtut waktu
Kondisi Akhir	Pengguna mendapatkan tampilan plot data atau *correlogram*

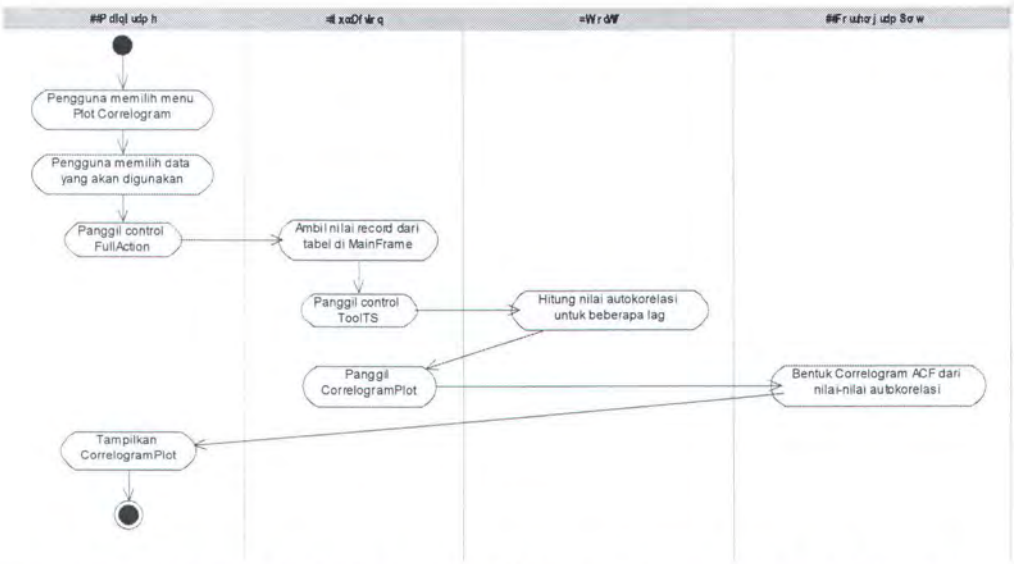
Tabel 4.2 Spesifikasi Use Case Menganalisis Stasioneritas Data (lanjutan)

Spesifikasi	Penjelasan
Aliran Aksi Normal	• Pengguna menekan tombol "Plot" pada panel data runtut waktu • Aplikasi menampilkan plot data atau • Pengguna memilih menu "Plot Correlogram" • Aplikasi menampilkan kotak dialog penggunaan data • Pengguna memilih data yang akan digunakan dari combo box • Aplikasi menampilkan *correlogram*

Aliran aksi normal *use case* ini menggunakan dua langkah dalam menganalisis stasioneritas data, yaitu dengan plot data dan dengan *correlogram*. Gambar 4.3 menunjukkan diagram aktivitas untuk *use case Menganalisis stasioneritas data* dengan plot data, sedangkan gambar 4.4 dengan *correlogram*.



Gambar 4.3 Diagram Aktivitas Use Case Menganalisis Stasioneritas Data dengan Plot Data



Gambar 4.4 Diagram Aktivitas Use Case Menganalisis Stasioneritas Data dengan Correlogram

4.1.3 Use Case Melakukan Differencing Terhadap Data Runtut Waktu

Use case ini merupakan proses differencing terhadap data runtut waktu yang dianggap tidak stasioner. Spesifikasi untuk use case ini ditunjukkan oleh tabel 4.3.

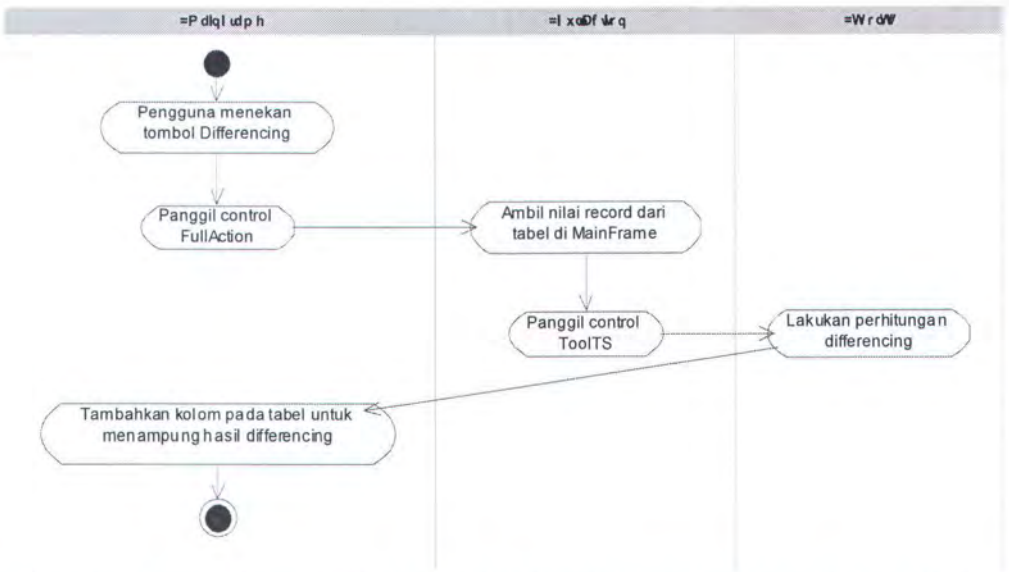
Tabel 4.3 Spesifikasi Use Case Melakukan Differencing Terhadap Data Runtut Waktu

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Memperoleh data runtut waktu yang stasioner
Ringkasan	Use case ini merupakan proses *differencing* terhadap data runtut waktu yang dianggap tidak stasioner
Masukan	• Nilai-nilai record data runtut waktu terpilih • Tingkat *differencing*
Luaran	Nilai-nilai hasil *differencing* terhadap record data runtut waktu terpilih

Tabel 4.3 Spesifikasi Use Case Melakukan Differencing Terhadap Data Runtut Waktu (lanjutan)

Spesifikasi	Penjelasan
Kondisi Awal	• Isi tabel pada panel data runtut waktu adalah nilai-nilai record data runtut waktu terpilih yang belum mengalami *differencing* atau • Isi tabel pada panel data runtut waktu adalah nilai-nilai record data runtut waktu terpilih termasuk hasil *differencing*
Kondisi Akhir	Tabel pada panel data runtut waktu menampilkan hasil *differencing* dari nilai-nilai record data runtut waktu terpilih
Aliran Aksi Normal	• Pengguna menekan tombol "Differencing" pada panel data runtut waktu • Aplikasi menambahkan sebuah kolom pada tabel di panel data runtut waktu yang berisi hasil *differencing*

Diagram aktivitas untuk use case Melakukan differencing terhadap data runtut waktu ini ditunjukkan oleh gambar 4.5.



Gambar 4.5 Diagram Aktivitas Use Case Melakukan Differencing Terhadap Data Runtut Waktu

4.1.4 Use Case Identifikasi Model

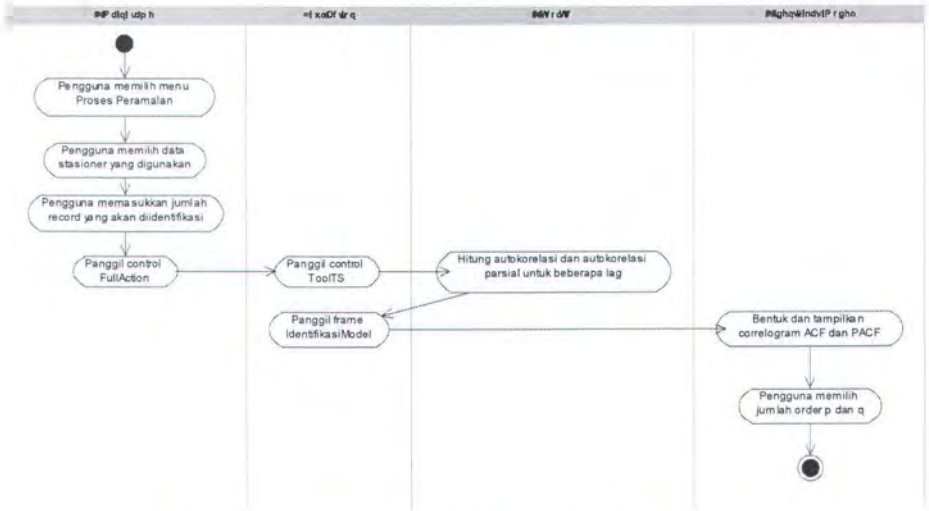
Use case ini merupakan proses pemilihan model ARIMA yang sesuai berdasarkan *correlogram* ACF dan PACF. Tabel 4.4 merupakan spesifikasi dari *use case* ini.

Tabel 4.4 Spesifikasi Use Case Identifikasi Model

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Memilih model ARIMA yang sesuai berdasarkan *correlogram* ACF dan PACF
Ringkasan	Use case ini merupakan proses pemilihan model ARIMA yang sesuai terhadap *n* record data runtut waktu stasioner terpilih berdasarkan *correlogram* ACF dan PACF
Masukan	• Data runtut waktu yang dianggap telah stasioner • Jumlah record data runtut waktu stasioner yang digunakan • Order AR (*p*)

	• Order I (*d*) • Order MA (*q*)
Luaran	Frame estimasi parameter untuk model yang dipilih
Kondisi Awal	Isi tabel pada panel data runtut waktu adalah nilai-nilai record data runtut waktu terpilih termasuk hasil *differencing*
Kondisi Akhir	Aplikasi menampilkan frame estimasi parameter untuk model yang dipilih
Aliran Aksi Normal	• Pengguna memilih menu "Proses Peramalan" • Aplikasi menampilkan kotak dialog "Pilih Data Stasioner" • Pengguna memilih data stasioner dari combo box di kotak dialog • Pengguna memasukkan jumlah record • Pengguna menekan tombol "OK" • Aplikasi menampilkan frame identifikasi model • Pengguna memilih jumlah order *p* dan *q* • Pengguna menekan tombol "Lanjutkan Estimasi Parameter"

Diagram aktivitas untuk *use case Identifikasi model* ini ditunjukkan oleh gambar 4.6.



Gambar 4.6 Diagram Aktivitas Identifikasi Model

4.1.5 Use Case Estimasi Parameter Model

Use case ini merupakan proses yang melakukan perhitungan dan iterasi untuk mendapatkan nilai konstanta dan koefisien dari model yang paling mendekati. Tabel 4.5merupakan spesifikasi dari *use case* ini.

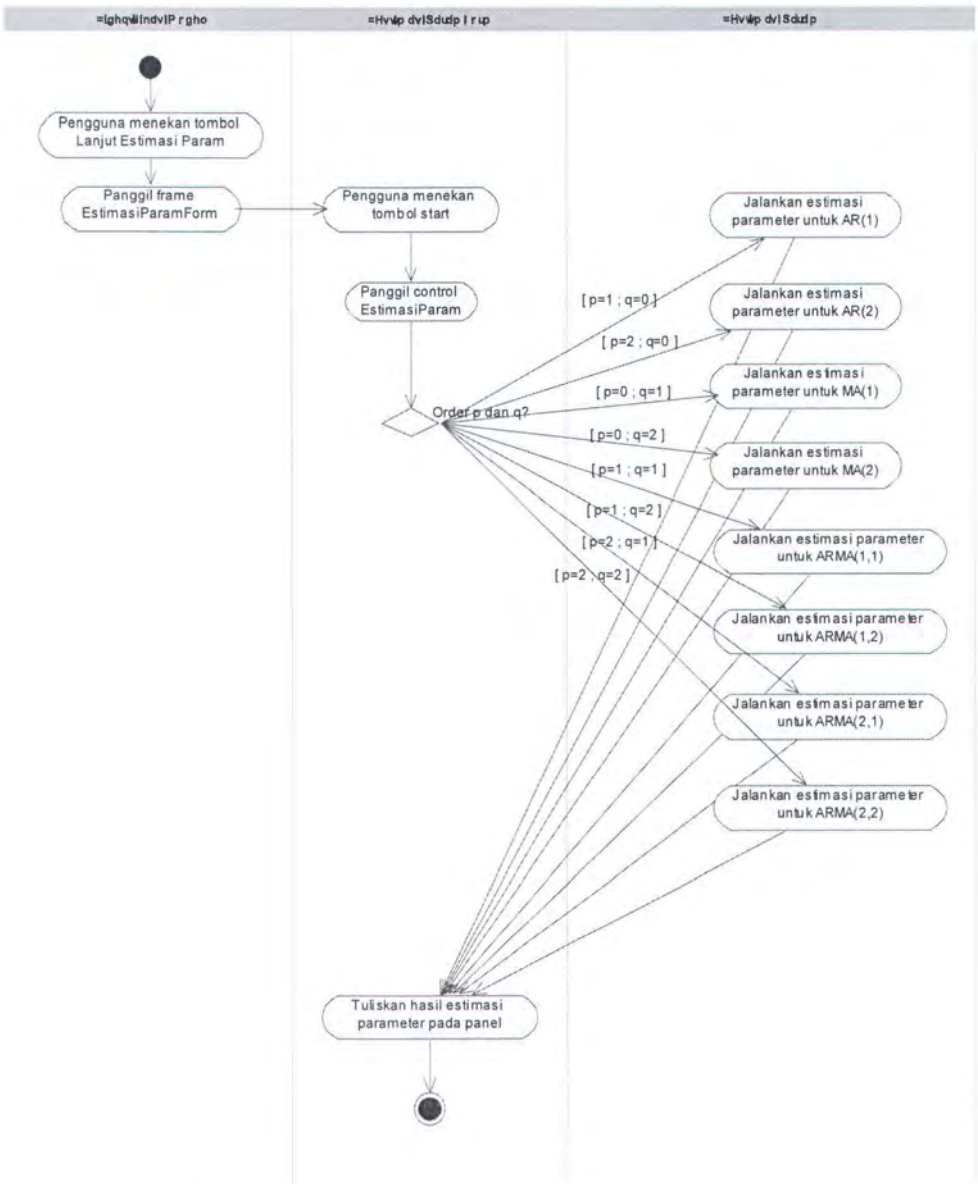
Tabel 4.5 Spesifikasi Use Case Estimasi Parameter Model

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Mendapatkan nilai konstanta dan koefisien dari model terpilih
Ringkasan	Use case ini merupakan proses yang melakukan perhitungan dan iterasi untuk mendapatkan nilai konstanta dan koefisien dari model yang paling mendekati
Masukan	• Nilai order *p*, *d*, dan *q* dari proses identifikasi model • Data runtut waktu stasioner sejumlah *n* record
Luaran	Konstanta dan koefisien untuk model terpilih
Kondisi Awal	Pengguna berada pada frame identifikasi model
Kondisi Akhir	• Pengguna berada pada frame estimasi parameter • Konstanta dan koefisien yang sesuai untuk model berhasil ditemukan

Tabel 4.5 Spesifikasi Use Case Estimasi Parameter Model (lanjutan)

Spesifikasi	Penjelasan
Aliran Aksi Normal	• Pengguna menekan tombol "Lanjutkan Estimasi Parameter" • Aplikasi menampilkan frame estimasi parameter untuk model • Pengguna menekan tombol "[start]" • Aplikasi melakukan perhitungan / iterasi untuk pencarian konstanta dan koefisien yang sesuai • Aplikasi menuliskan konstanta dan koefisien dari model yang sesuai pada panel

Aktivitas yang dilakukan dalam *use case* ini ditunjukkan oleh diagram aktivitas pada gambar 4.7. Estimasi parameter dilakukan untuk mendapatkan nilai konstanta dan koefisien dari model AR(1), AR(2), MA(1), MA(2), ARMA(1,1), ARMA(1,2), ARMA(2,1), dan ARMA(2,2). Sedangkan untuk komponen I, yang merupakan tingkat *differencing*, tidak dipengaruji dalam estimasi parameter ini, namun komponen *differencing* ini membedakan dalam hal penggunaan data.



Gambar 4.7 Diagram Aktivitas Estimasi Parameter Model

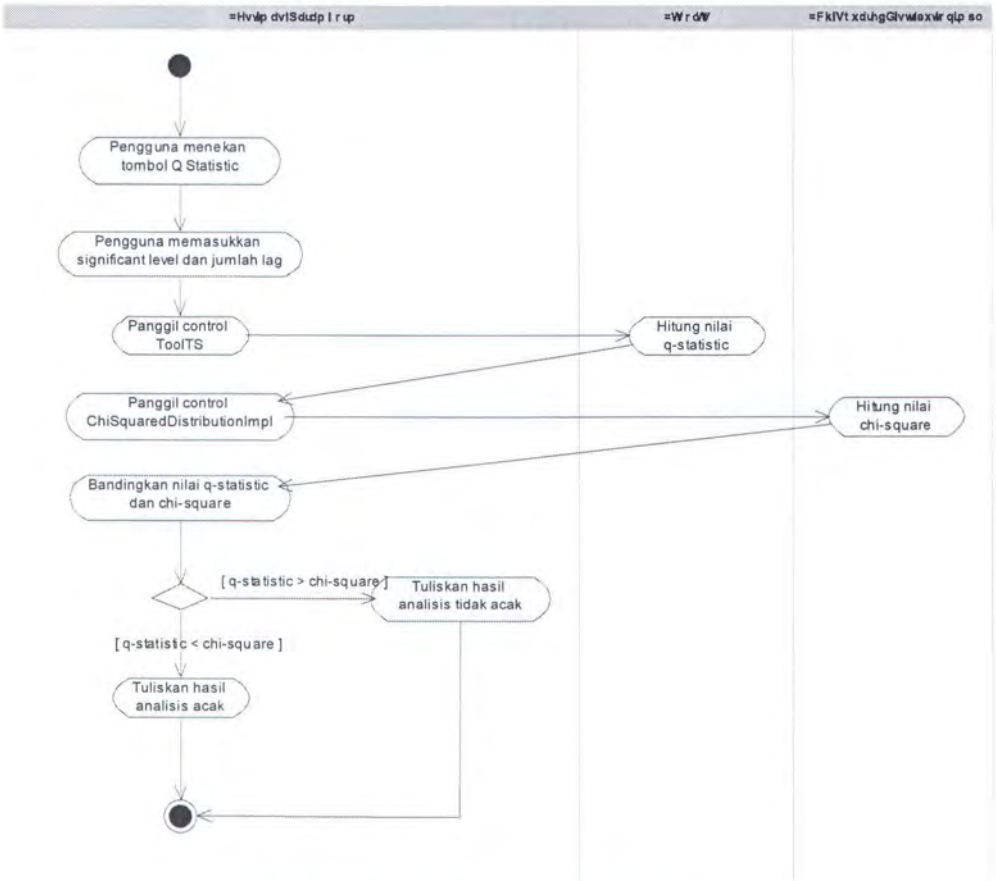
4.1.6 Use Case Diagnosis Model

Use case ini merupakan proses analisis menggunakan *Q-Statistic* terhadap residual atas model yang terbentuk, untuk menentukan keacakan residual yang diperoleh, yang juga mempengaruhi kelayakan model tersebut untuk digunakan. Tabel 4.6 merupakan spesifikasi untuk *use case* ini.

Tabel 4.6 Spesifikasi Use Case Diagnosis Model

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Melakukan analisis terhadap residual model
Ringkasan	Use case ini merupakan proses analisis (*Q-Statistic*) terhadap residual yang didapat atas model yang terbentuk
Masukan	• *Significant level* • Jumlah lag autokorelasi residual yang digunakan
Luaran	• Nilai *Q-Statistic* • Nilai *chi-square* • Nilai *Q-Statistic* dibandingkan dengan nilai *chi-square*
Kondisi Awal	• Pengguna berada pada frame estimasi parameter • Konstanta dan koefisien yang sesuai untuk model berhasil ditemukan
Kondisi Akhir	• Pengguna berada pada frame estimasi parameter • Aplikasi menuliskan keacakan dari residual model
Aliran Aksi Normal	• Pengguna menekan tombol "Q Statistic" • Aplikasi meminta masukan berupa *significant level* dan jumlah lag autokorelasi residual yang digunakan • Pengguna memasukkan *significant level* dan jumlah lag autokorelasi residual yang digunakan • Aplikasi menghitung nilai *Q-Statistic* dan *chi-square* • Aplikasi membandingkan nilai *Q-Statistic* dengan *chi-square* • Aplikasi menuliskan hasil perbandingan

Aktivitas yang terjadi dalam *use case Diagnosis model* ini ditunjukkan oleh diagram aktivitas pada gambar 4.8.



Gambar 4.8 Diagram Aktivitas Diagnosis Model

4.1.7 Use Case Menjalankan Peramalan Permintaan

Use case ini merupakan proses melakukan peramalan untuk beberapa periode ke depan dengan model yang didapat. Tabel 4.7 menunjukkan spesifikasi *use case* ini.

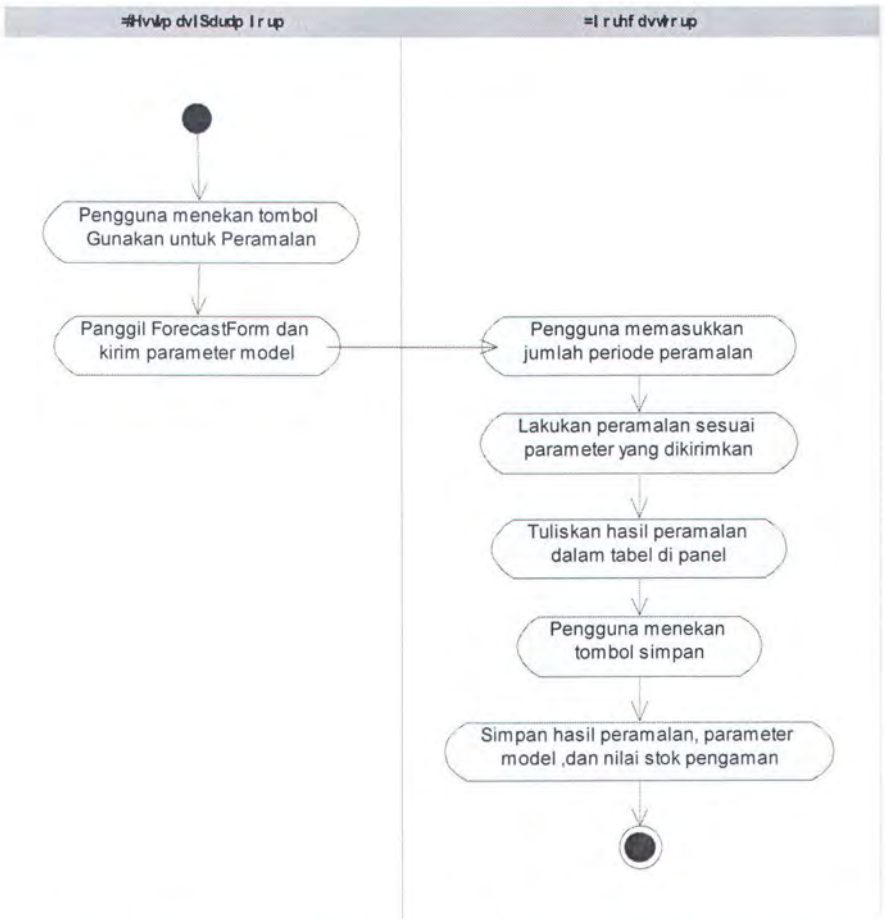
Tabel 4.7 Spesifikasi Use Case Menjalankan Peramalan Permintaan

Spesifikasi	Penjelasan
Aktor	Pengguna
Tujuan	Mendapatkan hasil peramalan atas data runtut waktu terpilih menggunakan model yang didapat
Ringkasan	Use case ini melakukan proses peramalan terhadap data runtut waktu terpilih dengan model yang didapat

Tabel 4.7 Spesifikasi Use Case Menjalankan Peramalan Permintaan (lanjutan)

Spesifikasi	Penjelasan
Masukan	• Jumlah periode yang akan diramalkan • Persamaan model yang didapat
Luaran	Hasil peramalan untuk *t* periode
Kondisi Awal	• Pengguna berada pada frame estimasi parameter • Konstanta dan koefisien yang sesuai untuk model berhasil ditemukan
Kondisi Akhir	• Pengguna berada pada frame peramalan data runtut waktu • Hasil peramalan untuk *t* periode berhasil didapatkan
Aliran Aksi Normal	• Pengguna menekan "Gunakan untuk Peramalan" pada frame estimasi parameter • Aplikasi menampilkan frame peramalan • Pengguna memasukkan jumlah periode peramalan • Pengguna menekan tombol "ramal" • Aplikasi menampilkan nilai-nilai hasil peramalan pada frame peramalan

Gambar 4.9 merupakan diagram aktivitas dari *use case Menjalankan peramalan permintaan.*



Gambar 4.9 Diagram Aktivitas Use Case Menajalankan Peramalan Permintaan

4.1.8 Use Case Menjalankan Perhitungan Perencanaan Produksi

Use case ini merupakan proses optimasi untuk mendapatkan jumlah biaya lembur dan biaya penyimpanan persediaan minimum. Spesifikasi dari *use case* ini dituliskan pada tabel 4.8.

Tabel 4.8 Spesifikasi Use CaseMenjalankan Perhitungan Perencanaan Produksi

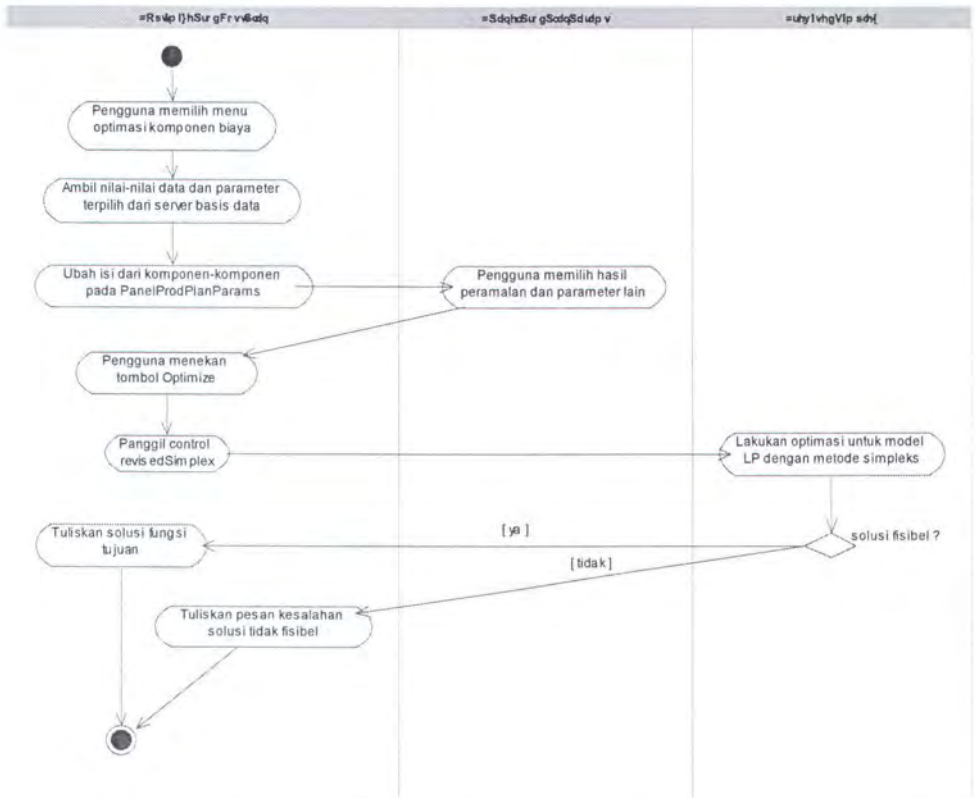
Spesifikasi	Penjelasan
Aktor	• Pengguna • Server Basis Data
Tujuan	Mendapatkan jumlah biaya lembur dan biaya penyimpanan persediaan minimum hasil optimasi

Tabel 4.8 Spesifikasi Use CaseMenjalankan Perhitungan Perencanaan Produksi (lanjutan)

Spesifikasi	Penjelasan
Ringkasan	Use case ini merupakan proses optimasi untuk mendapatkan jumlah biaya lembur dan biaya penyimpanan persediaan minimum
Masukan	• Jumlah periode • Jumlah jenis produk • Jenis produk • Biaya penyimpanan per satuan • Produktivitas per satuan • Hasil ramalan dan stok pengaman • Biaya lembur per satuan • Waktu kerja reguler • Waktu lembur maksimum • Jumlah persediaan awal
Luaran	Biaya lembur dan biaya penyimpanan persediaan minimum
Kondisi Awal	• Data dan parameter produksi telah ada dalam server basis data • Hasil peramalan telah tersimpan dalam server basis data
Kondisi Akhir	Biaya lembur dan biaya penyimpanan persediaan minimum berhasil dioptimasi
Aliran Aksi Normal	• Pengguna memilih menu "Optimasi Komponen Biaya" • Aplikasi menampilkan frame optimasi biaya • Pengguna memilih data dan parameter yang akan digunakan dari combo box pada frame optimasi biaya • Pengguna menekan tombol "Gunakan" • Aplikasi mengubah isi tabel biaya penyimpanan, produktivitas, hasil peramalan, stok pengaman,

	persediaan awal, biaya lembur • Pengguna memilih hasil peramalan untuk masing-masing produk yang bersangkutan • Pengguna memilih hasil perhitungan stok pengaman untuk masing-masing produk yang bersangkutan • Pengguna menekan tombol "Optimize" • Aplikasi melakukan optimasi dengan metode simpleks • Aplikasi mendapatkan nilai optimal dari fungsi tujuan yang diharapkan berupa jumlah minimum biaya lembur dan biaya penyimpanan persediaan

Aktivitas yang dilakukan oleh *use case Menjalankan perhitungan perencanaan produksi* ini ditunjukkan dalam diagram aktivitas pada gambar 4.10.



Gambar 4.10 Diagram Aktivitas Use Case Menjalankan Perhitungan Perencanaan Produksi

4.2 Desain Aplikasi

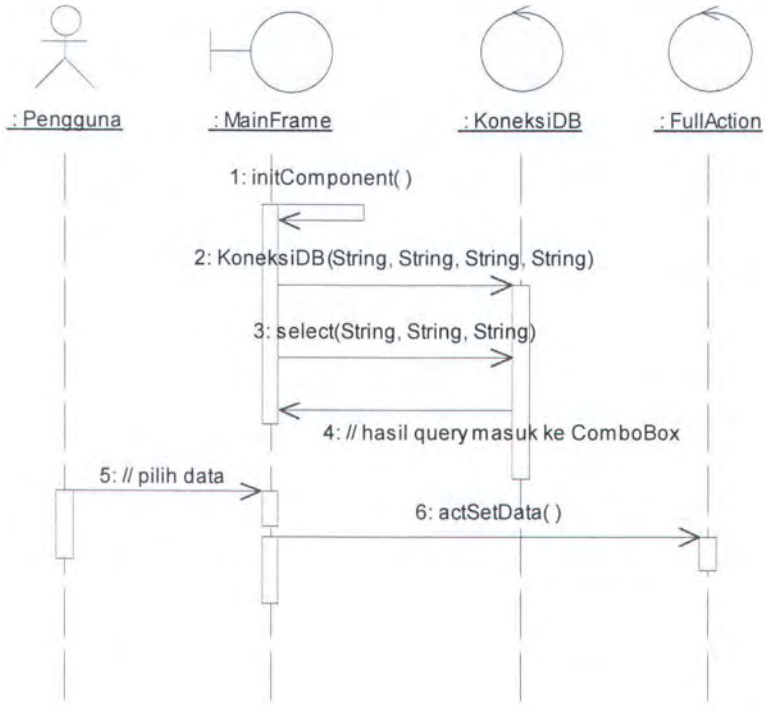
Setelah proses-proses yang akan dilakukan oleh aplikasi diketahui melalui diagram *use case*, kemudian aktivitas-aktivitas dari setiap *use case* tersebut juga telah dispesifikasikan, tahap selanjutnya adalah membuat desain dari aplikasi itu sendiri, yaitu berupa realisasi dari *use case* dilanjutkan desain dari tiap class penyusunnya.

4.2.1 Realisasi Use Case

Realisasi use case dari aplikasi menggunakan diagram *sequence* untuk menjelaskan alur detil dari setiap *use case* yang ada. Diagram *sequence* menggambarkan hubungan antar objek berdasarkan urutan waktu.

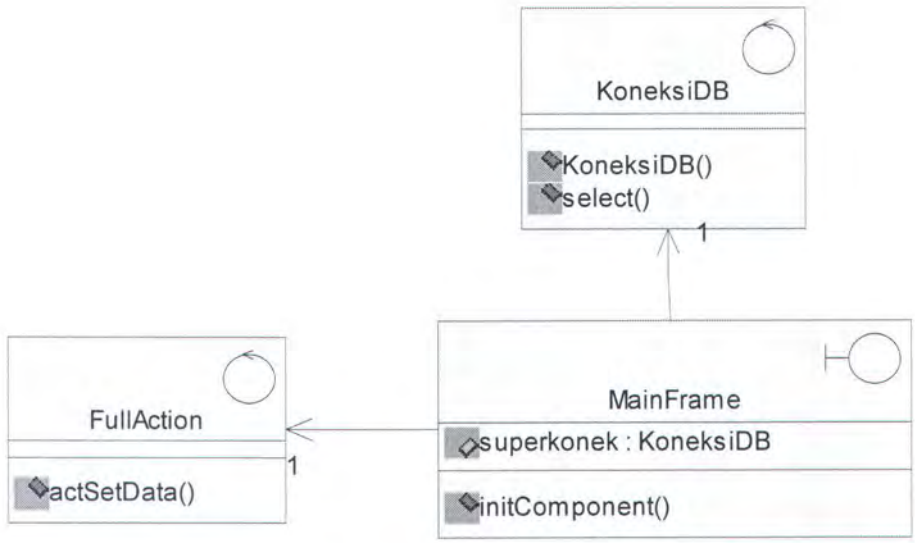
4.2.1.1 Memilih Data Runtut Waktu

Pada *use case Memilih data runtut waktu*, pengguna memilih data runtut waktu yang akan digunakan sebagai data yang akan dianalisis untuk membuat model ARIMA. Diagram *sequence* dari use case ini digambarkan oleh gambar 4.11.



Gambar 4.11 Diagram Sequence Use Case Memilih Data Runtut Waktu

Dari diagram *sequence* pada gambar 4.11 yang menggambarkan proses dalam *use case Memilih Data Runtut Waktu*, terdapat beberapa objek dari tiga class yang terlibat yaitu `MainFrame`, `KoneksiDB`, dan `FullAction`. Partisipasi setiap objek tersebut digambarkan pada diagram class pada gambar 4.12.



Gambar 4.12 VOPC Memilih Data Runtut Waktu

Penjelasan aliran operasi atau pesan (*message*) pada diagram sequence tersebut dituliskan dalam tabel 4.9.

Tabel 4.9 Penjelasan Aliran Operasi / Message Use Case Memilih Data Runtut Waktu

No.	Operasi / Message	Penjelasan
1	initComponent()	Termasuk didalamnya pendefinisian class koneksi dengan basis data
2	KoneksiDB(String,String, String,String)	Konstruktor class KoneksiDB, mendefinisikan koneksi aplikasi dengan basis data. Parameter String yang dibawa berturut-turut adalah lokasi server, nama basis data, *username*, *password*
3	select(String,String, String)	Salah satu method dari class KoneksiDB yang memberikan pengembalian berupa hasil kueri `select` pada basis data.
4	// hasil query masuk ke ComboBox	Hasil dari kueri dimasukkan dalam pilihan pada komponen ComboBox
5	// pilih data	Pemilihan data oleh pengguna dari ComboBox

Tabel 4.9 Penjelasan Aliran Operasi / Message Use Case Memilih Data Runtut Waktu (lanjutan)

No.	Operasi / Message	Penjelasan
6	actSetData()	Dijalankan untuk mengubah isi tabel di `MainFrame`

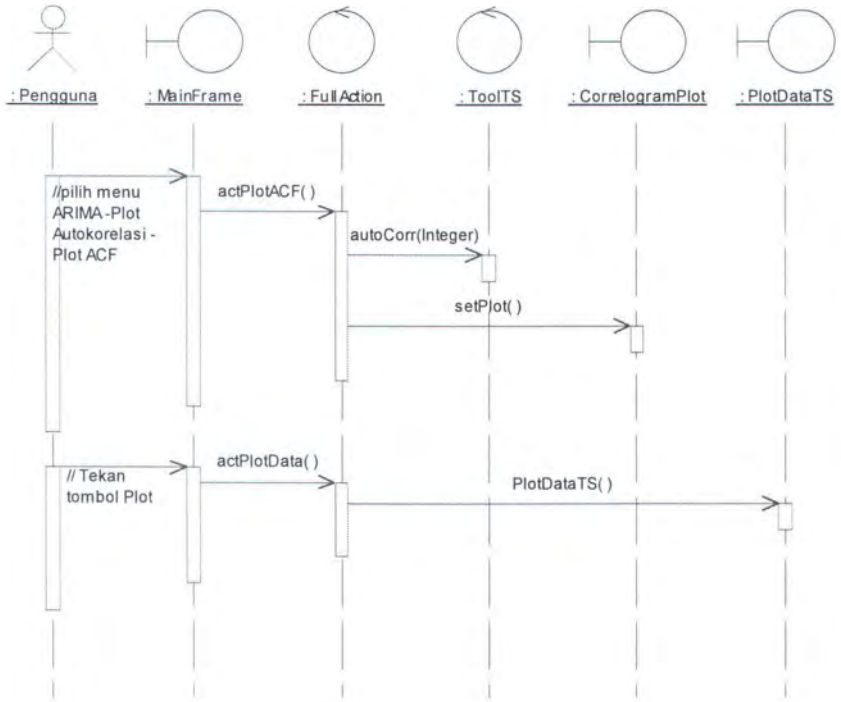
Diagram *sequence* pada gambar 4.12, dimulai dari class `MainFrame` mendefinisikan komponen-komponennya termasuk komponen `ComboBox` yang memuat daftar data runtut waktu. Daftar data runtut waktu yang ditampilkan pada `ComboBox` diperoleh dari database, yang dipanggil oleh objek yang memiliki class `KoneksiDB` menggunakan method `select()`. Setelah pengguna memilih data runtut waktu pilihannya, maka oleh method `actSetData()` tabel akan menampilkan data runtut waktu terpilih tersebut.

4.2.1.2 Menganalisis Stasioneritas Data

Sebelum data diidentifikasi untuk dicari model data yang tepat, maka data observasi harus dalam kondisi stasioner. Untuk memenuhi kondisi itu, data perlu diperiksa dengan analisis terhadap plot data maupun plot fungsi autokorelasinya. Gambar 4.13 adalah diagram *sequence* dari proses ini.

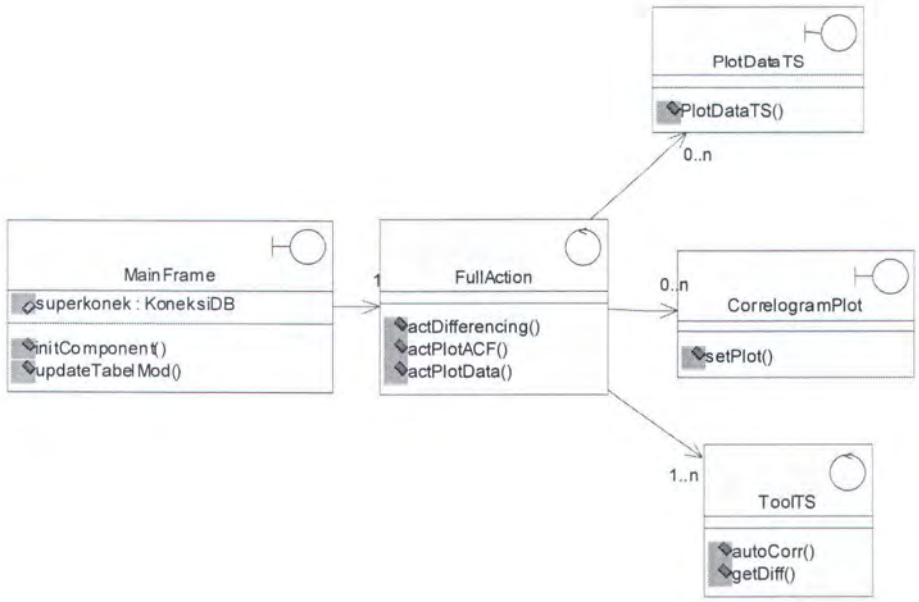
Untuk melihat plot fungsi autokorelasi dari data yang terpilih, pengguna memilih menu "ARIMA – Plot Autokorelasi – Plot ACF". Sesaat setelah pengguna memilih menu tersebut, class `FullAction` akan memanggil method `actPlotACF()`, yang selanjutnya akan menjalankan method `autoCorr()` milik class `ToolTS`, yang memiliki nilai pengembalian berupa nilai koefisien autokorelasi untuk lag tertentu. Hasil perhitungan koefisien autokorelasi untuk setiap lag ditampilkan dalam bentuk *correlogram* yang dilakukan oleh class `CorrelogramPlot` dengan method `setPlot()`.

Plot data asli maupun data hasil *differencing* dapat ditampilkan dengan menekan tombol "Plot". Plot data akan ditampilkan oleh class `PlotDataTS` dengan memanggil konstruktornya.



Gambar 4.13 Diagram Sequence Use Case Menganalisis Stasioneritas Data

Beberapa objek dari class `MainFrame`, `FullAction`, `ToolTS`, `PlotDataTS`, dam `CorrelogramPlot` terlibat dalam proses pada *use case Menganalisis Stasioneritas Data*. Partisipasi dari lima class tersebut digambarkan pada VOPC pada gambar 4.14.



Gambar 4.14 VOPC Menganalisis Stasioneritas Data

Penjelasan mengenai operasi maupun message pada diagram *sequence* di gambar 4.14 ditunjukkan oleh tabel 4.10.

Tabel 4.10 Penjelasan Operasi/Message Use Case Menganalisis Stasioneritas Data

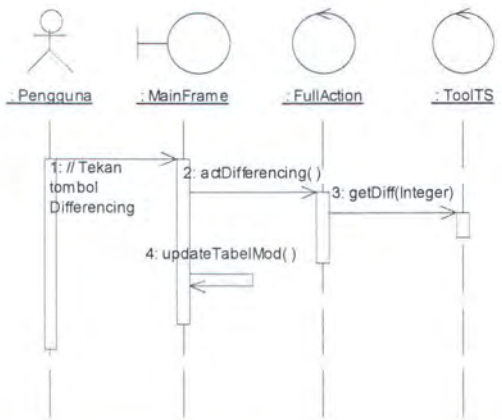
No.	Operasi / Message	Penjelasan
1	// pilih menu ARIMA-Plot Autokorelasi-PlotACF	Dilakukan oleh pengguna untuk memulai proses analisis stasioneritas
2	actPlotACF()	Dijalankan untuk membuat *correlogram* ACF dari nilai sekumpulan nilai koefisien autokorelasi.
3	autoCorr(Integer)	Dilakukan untuk mendapatkan nilai koefisien autokorelasi untuk lag tertentu. Parameter Integer merupakan nilai lag
4	setPlot()	Dilakukan untuk menggambar plot dari *correlogram*
5	// tekan tombol plot	Dilakukan pengguna untuk melihat plot data
6	actPlotData()	Dijalankan untuk membuat plot dari data

Tabel 4.10 Penjelasan Operasi/Message Use Case Menganalisis Stasioneritas Data (lanjutan)

No.	Operasi / Message	Penjelasan
7	PlotDataTS()	Dijalankan untuk menggambar plot data

4.2.1.3 Melakukan Differencing Terhadap Data Runtut Waktu

Differencing dilakukan apabila data yang akan digunakan dalam peramalan belum stasioner. Diagram *sequence* dari *use case* ini digambarkan oleh gambar 4.15.



Gambar 4.15 Diagram Sequence Use Case Melakukan Differencing

Penjelasan dari operasi atau *message* yang terdapat pada diagram *sequence* tersebut dituliskan pada tabel 4.11.

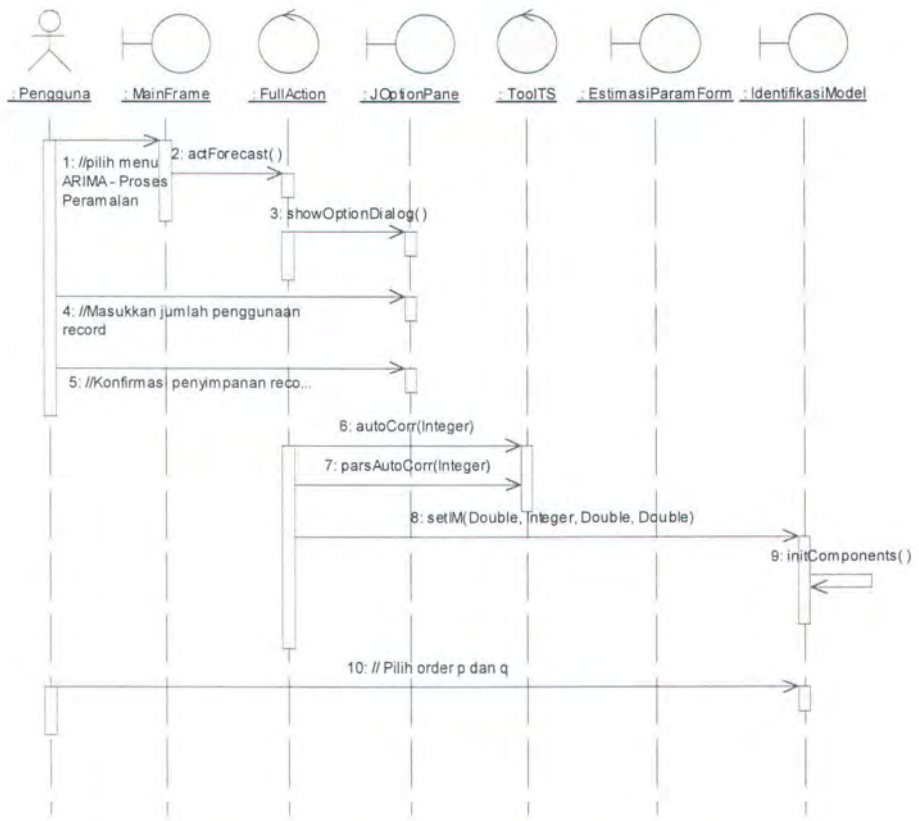
Tabel 4.11 Penjelasan Operasi/Message Use Case Melakukan Differencing

No.	Operasi / Message	Penjelasan
1	// tekan tombol *differencing*	Dilakukan oleh pengguna untuk mendapatkan nilai *differencing*
2	actDifferencing()	Dijalankan untuk memulai proses differencing
3	getDiff(Integer)	Dilakukan untuk mendapatkan nilai hasil perhitungan *differencing* dengan tingkat tertentu
4	updateTabelMod()	Dilakukan untuk mengubah isi dari tabel

		pada `MainFrame` dengan menambah kolom yang berisi record hasil *differencing*

4.2.1.4 Identifikasi Model

Identifikasi model dilakukan pengguna dengan memilih model ARIMA yang sesuai berdasarkan *correlogram* ACF dan PACF. Diagram *sequence* untuk *use case* ini digambarkan pada gambar 4.16.



Gambar 4.16 Diagram Sequence Use Case Identifikasi Model

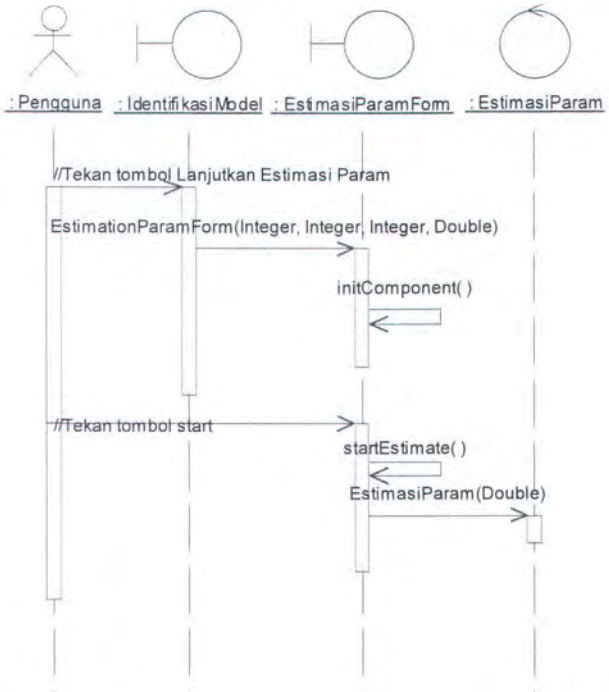
Penjelasan dari operasi dan *message* yang terdapat pada diagram *sequence* tersebut dituliskan pada tabel 4.12.

Tabel 4.12 Penjelasan Operasi/Message Use Case Identifikasi Model

No.	Operasi / Message	Penjelasan
1	// pilih menu ARIMA-Proses Peramalan	Dilakukan oleh pengguna untuk memulai proses peramalan, yang diawali dengan identifikasi model
2	actForecast()	Dijalankan untuk melakukan rangkai proses peramalan
3	showOptionDialog()	Dilakukan untuk meminta jumlah penggunaan record
4	// masukkan jumlah penggunaan record	Masukan oleh pengguna berupa jumalh record yang akan digunakan
5	// konfirmasi penyimpanan record sisa	Persetujuan penyimpanan record sisa sebagai data real
6	autoCorr(Integer)	Menghitung koefisien autokorelasi
7	parsAutoCorr(Integer)	Menghitung koefisien autokorelasi parsial
8	setIM(Double,Integer,Double,Double)	Mengatur parameter data, lag nilai-nilai autokorelasi dan autokorelasi parsial
9	initComponent()	Menginisialisasi frame identifikasi
10	// pilih order p dan q	Dilakukan pengguna dalam menetapkan nilai order p dan q

4.2.1.5 Estimasi Parameter Model

Estimasi parameter model dilakukan oleh sebuah class *control* yang dikhususkan untuk menangani estimasi parameter, dan pengembalian yang dapat diperoleh dari class tersebut adalah nilai-nilai konstanta maupun koefisien sesuai dengan model ARIMA yang dipilih. Diagram *sequence* untuk *use case* ini ditunjukkan oleh gambar 4.17.



Gambar 4.17 Diagram Sequence Estimasi Parameter Model

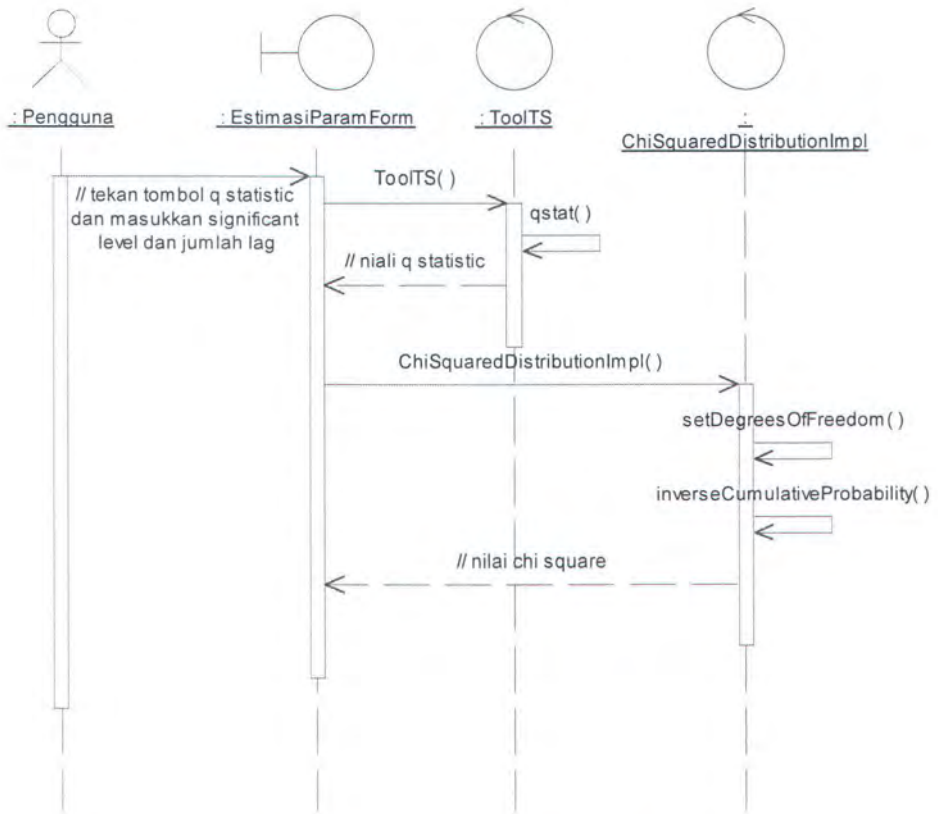
Penjelasan dari operasi dan *message* pada diagram *sequence* ini dijelaskan oleh tabel 4.13.

Tabel 4.13 Penjelasan Operasi/Message Use Case Estimasi Parameter Model

No.	Operasi / Message	Penjelasan
1	// tekan tombol Lanjutkan Estimasi Param	Dilakukan oleh pengguna untuk melanjutkan pada proses estimasi parameter
2	EstimationParamForm(Integer, Integer,Integer,Double)	Konstruktor class EstimationParamForm dengan parameter order AR,order MA, tingkat differencing, dan data.
3	initComponent()	Inisialisasi class EstimationParamForm
4	// tekan tombol start	Dilakukan untuk memulai proses estimasi
5	startEstimate()	Memanggil class control EstimasiParam
6	EstimasiParam(Double)	Konstruktor class EstimasiParam dengan parameter berupa data

4.2.1.6 Diagnosis Model

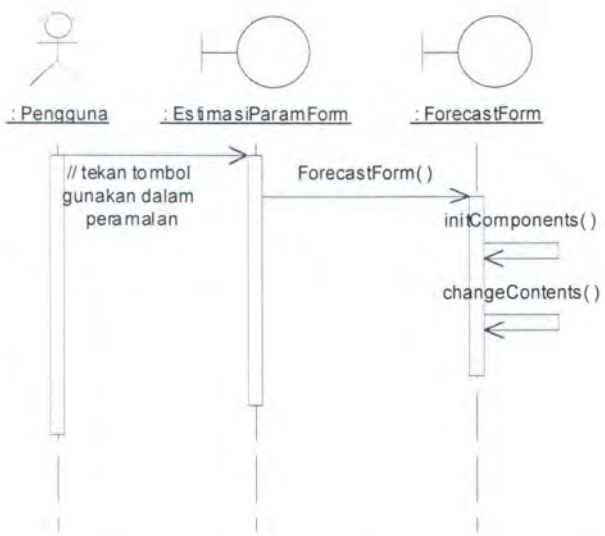
Use case ini menggunakan dua class *control* utama yaitu `ToolTS` dan `ChiSquaredDistributionImpl` untuk mendapatkan nilai *Q-Statistic* dan nilai *chi square*. Gambar 4.18 menunjukkan diagram *sequence* untuk *use case Diagnosis model* ini.



Gambar 4.18 Diagram Sequence Use Case Diagnosis Model

4.2.1.7 Menjalankan Peramalan Permintaan

Use case Menjalankan peramalan permintaan memberikan hasil berupa hasil ramalan. Diagram sequence untuk use case ini ditunjukkan oleh gambar 4.19.



Gambar 4.19 Diagram Sequence Use Case Menjalankan Peramalan Permintaan

Setelah mendapatkan nilai parameter yang diinginkan, pengguna dapat menggunakan nilai-nilai parameter itu dalam proses peramalan dengan menekan tombol "Gunakan dalam Peramalan". Proses peramalan dilakukan untuk beberapa periode ke depan, yang ditentukan oleh pengguna dengan mengisi *textfield* yang terdapat di frame `ForecastForm`. Hasil akhir dari proses ini adalah nilai-nilai hasil peramalan untuk beberapa periode ke depan, serta nilai-nilai stok pengaman (*safety stock*) yang akan digunakan dalam proses *Perhitungan Perencanaan Produksi*.

4.2.1.8 Menjalankan Perhitungan Perencanaan Produksi

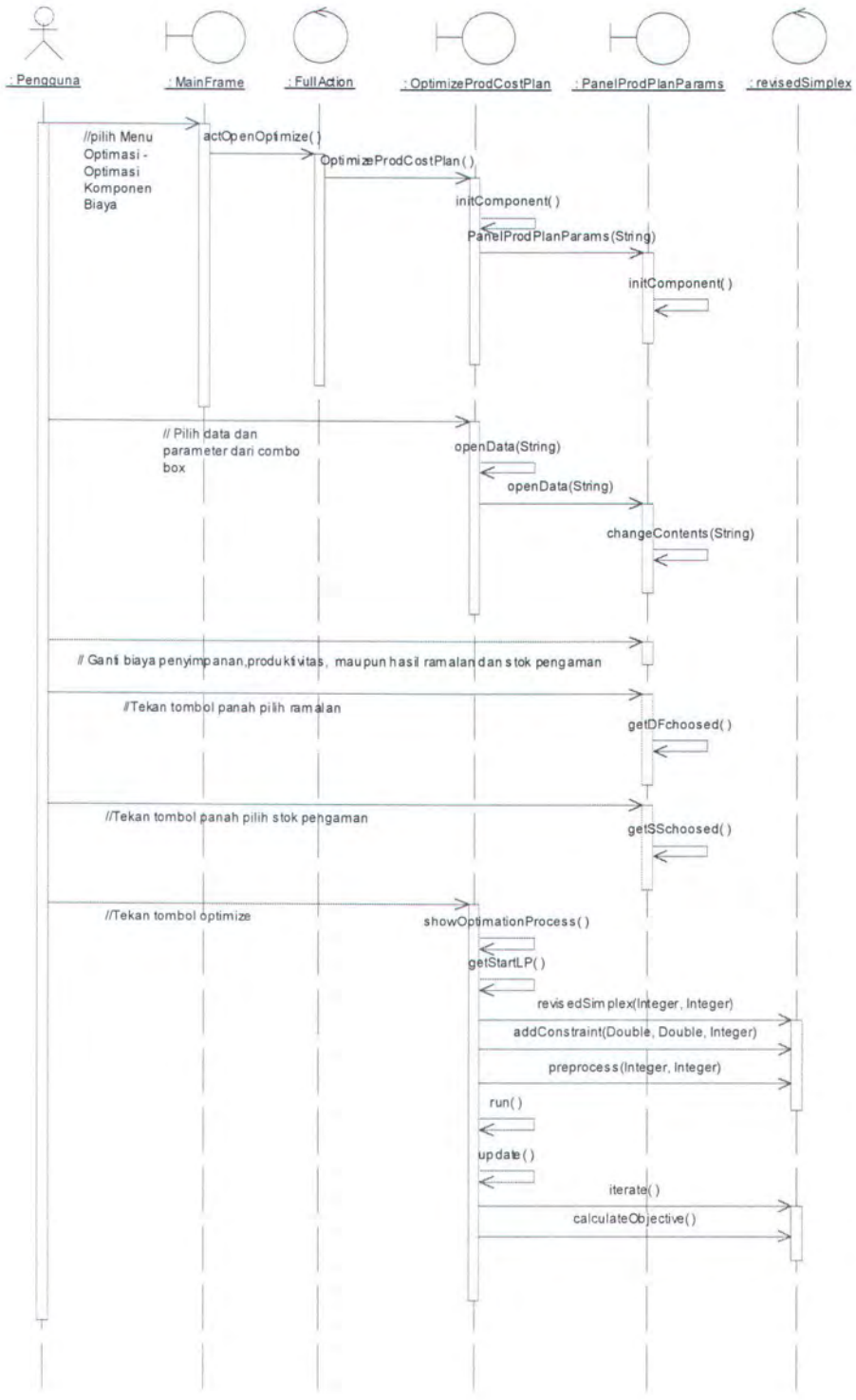
Nilai-nilai yang didapat dari hasil peramalan selanjutnya digunakan untuk melakukan perencanaan produksi. Perencanaan produksi yang dilakukan disini adalah perhitungan total biaya lembur dan biaya penyimpanan yang timbul selama proses produksi. Gambar 4.20 merupakan diagram *sequence* yang menjelaskan proses ini.

Proses dimulai dengan pemilihan menu "Optimasi – Optimasi Komponen Biaya" pada `MainFrame` oleh pengguna. Selanjutnya aplikasi akan menampilkan frame `OptimizeProdCostPlan` yang menginisialisasi panel `PanelProdPlanParams` di dalamnya.

Pengguna memilih data dan parameter yang akan digunakan dari `ComboBox cbGetProdParams`, untuk mengubah isi dari panel `PanelProdPlanParams`. Isi dari komponen-komponen pada panel tersebut berubah sesuai dengan data dan parameter yang dipilih. Komponen-komponen itu adalah tabel biaya penyimpanan, tabel produktivitas, dan tabel inventori awal, serta label jumlah periode perencanaan (T), label jumlah jenis produk (N) dan *textfield* biaya lembur.

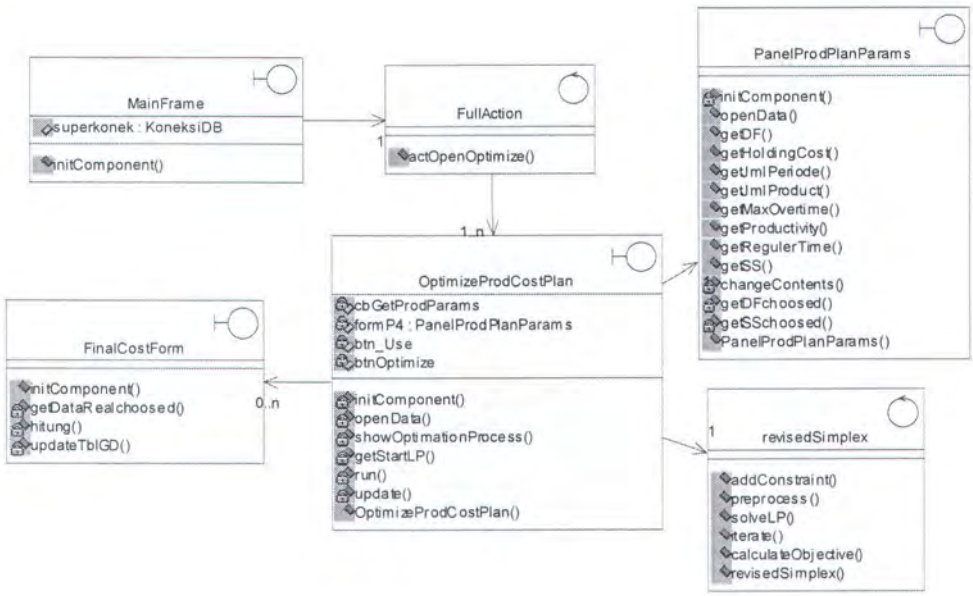
Selanjutnya pengguna memilih hasil ramalan dan stok pengaman yang akan digunakan dari tabel hasil ramalan permintaan dan tabel hasil perhitungan stok pengaman. Dilanjutkan dengan menekan tombol yang berada di samping masing-masing tabel tersebut untuk membawa nilai-nilai dari tabel tersebut ke dalam tabel yang berada di bawahnya yaitu `tabelGD`, melalui method `getDFchoosed()` dan method `getSSchoosed()`.

Proses dilanjutkan dengan mencari nilai optimum dari fungsi tujuan, yaitu nilai minimum dari total jumlah biaya penyimpanan dan jumlah biaya lembur. Pengguna menekan tombol "optimize" untuk memanggil method `showOptimationProcess()`, yang dilanjutkan dengan method `getStartLP()`. Dalam method `getStartLP()`, class `OptimizeProdCostPlan` akan memanggil objek dengan class `revisedSimplex`, yang dalam hal ini bertanggung jawab untuk melakukan proses optimasi dengan pemrograman linier. Class `revisedSimplex` akan memanggil method-method `addConstraint()`, `preprocess()`, `iterate()`, dan `calculateObjective()`, yang berturut-turut merupakan proses pendefinisian pembatas-pembatas, persiapan tabel Simplex, perulangan-perulangan, hingga didapatkan nilai optimum fungsi tujuan.



Gambar 4.20 Diagram Sequence Menjalankan Perhitungan Perencanaan Produksi

Untuk proses dari *use case Menjalankan Perhitungan Perencanaan Produksi*, terdapat enam class yang terlibat yaitu `MainFrame`, `FullAction`, `OptimizeProdCostPlan`, `PanelProdPlanParams`, `revisedSimplex`, dan `FinalCostForm`. Partisipasi setiap class itu digambarkan dalam VOPC pada gambar 4.21.

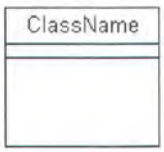


Gambar 4.21 VOPC Menjalankan Perhitungan Peramalan Produksi

4.2.2 Desain Class Global

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (method/fungsi). Diagram class menggambarkan eksistensi dari suatu class-class dan hubungan diantara mereka dalam desain suatu sistem. Sebuah diagram class dapat merepresentasikan semua atau sebagian struktur dari suatu sistem.

Dalam desain model, class digambarkan sebagai kotak yang terdiri atas tiga bagian. Gambar 4.22 adalah gambaran dari sebuah class.



Gambar 4.22 Class

Dari gambar 4.22, penjelasan dari ketiga bagian tersebut adalah sebagai berikut :

a. Nama class

Nama class terletak pada bagian teratas. Nama class merupakan identitas dari suatu class. Umumnya class juga memiliki *stereotype*, yang menggambarkan sifat class tersebut. *Stereotype* dari class antara lain adalah *boundary*, *control*, dan *entity*.

Boundary class merupakan antarmuka antara sistem dengan lingkungan sekitar sistem (seseorang atau sistem lain). *Control* class adalah class yang melakukan suatu hal yang spesifik. Sedangkan *entity* class adalah class yang digunakan untuk memodelkan suatu informasi dan menyimpannya.

b. Daftar atribut

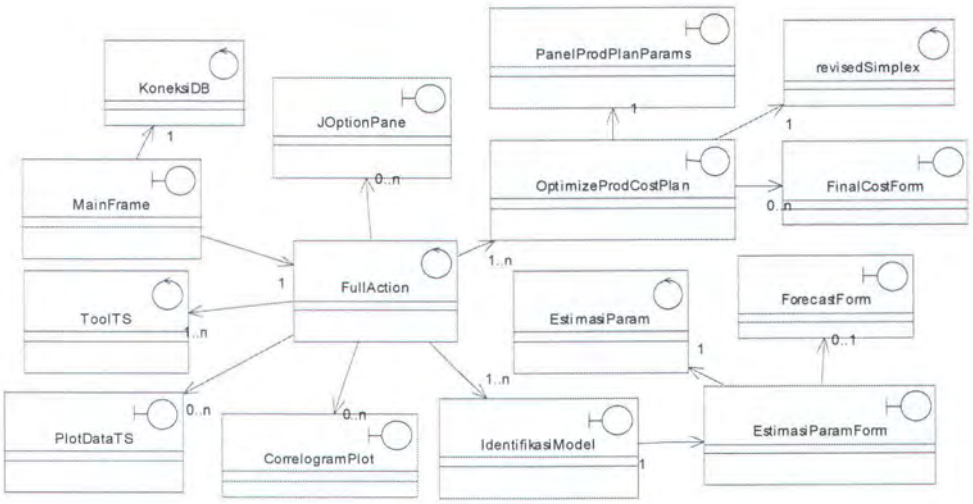
Daftar atribut terletak pada bagian tengah. Atribut mendefinisikan karakteristik dari suatu class. Sebuah objek dari suatu class dimungkinkan memiliki atribut-atribut yang sama, namun memiliki nilai yang berbeda.

c. Daftar operasi (method)

Daftar operasi (method) terletak pada bagian bawah. Objek dari suatu class melakukan suatu manipulasi melalui operasi (method) yang dimilikinya.

Untuk setiap proses yang digambarkan oleh diagram *sequence* pada sub bab 4.2.2, akan melibatkan beberapa objek dari class. Gambaran partisipasi setiap class yang terlibat, *View of Participated Class* (VOPC), dalam masing-masing diagram *sequence* tersebut dapat dilakukan secara parsial, yang untuk selanjutnya digambarkan secara global.

Secara global, partisipasi dari setiap class pada sistem ini digambarkan pada diagram class gambar 4.23. Diagram class ini terdiri dari lima belas class, yang secara global memiliki hubungan partisipasi seperti pada gambar 4.23 tersebut.

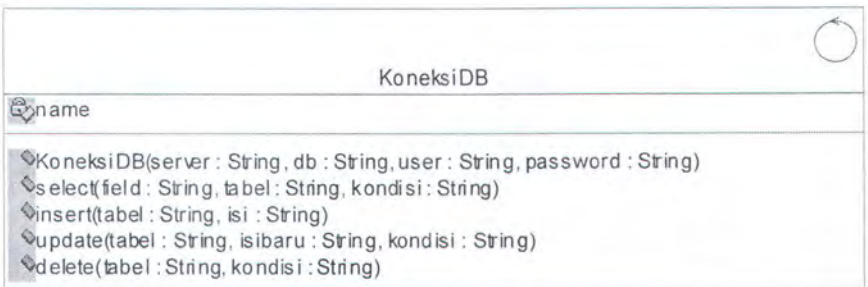


Gambar 4.23 Diagram Class Global

Kelima belas class pada diagram class tersebut memiliki tanggung jawab dan fungsi yang spesifik pada sistem. Penjelasan mengenai setiap class tersebut adalah sebagai berikut :

4.2.2.1 Class KoneksiDB

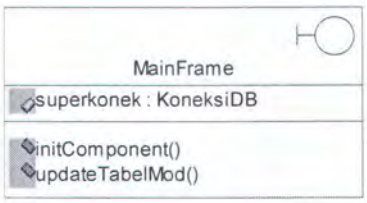
Class ini bertanggungjawab dalam menangani koneksi antara aplikasi dengan database dan menjalankan kueri `select`, `insert`, `update`, dan `delete`. Gambar 4.24 adalah gambar class `KoneksiDB` beserta atribut dan operasi yang dimilikinya.



Gambar 4.24 Class KoneksiDB

4.2.2.2 Class MainFrame

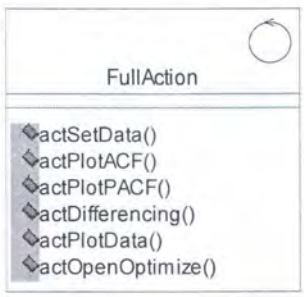
Class ini diturunkan dari class `JFrame` dari package `javax.swing`. Berfungsi sebagai antarmuka utama dari aplikasi dan memberikan menu-menu yang menghubungkan fitur-fitur aplikasi dengan pengguna. Class beserta atribut dan operasi yang dimilikinya digambarkan pada gambar 4.25.



Gambar 4.25 Class MainFrame

4.2.2.3 Class FullAction

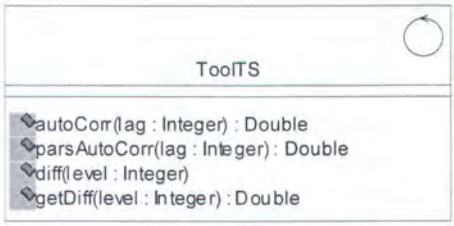
Class ini menampung perintah-perintah yang dipanggil oleh class `MainFrame`. Berfungsi sebagai penghubung antara class `MainFrame` dengan class-class lain, selain `KoneksiDB`. Gambar 4.26 menggambarkan class ini.



Gambar 4.26 Class FullAction

4.2.2.4 Class ToolTS

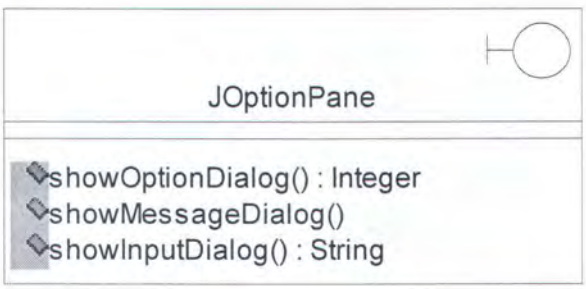
Class ini berperan sebagai alat untuk mengolah data runtut waktu, seperti melakukan *differencing*, perhitungan autokorelasi, dan perhitungan autokorelasi parsial.Gambaran mengenai class ini tampak pada gambar 4.27.



Gambar 4.27 Class ToolTS

4.2.2.5 Class JOptionPane

Class ini digunakan untuk menampilkan kotak dialog baik berupa pesan informasi, pesan konfirmasi, isian, maupun kotak dialog lainnya. Class `JOptionPane` berada pada package `javax.swing`. Gambar 4.28 menggambarkan class `JOptionPane`.

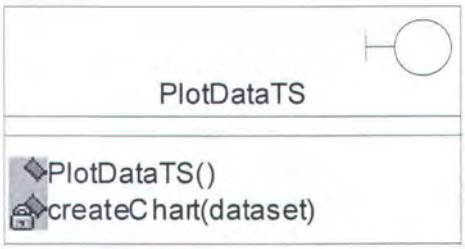


Gambar 4.28 Class JOptionPane

Sebenarnya class ini memiliki jumlah daftar atribut dan operasi yang lebih banyak daripada jumlah daftar atribut dan operasi yang digambarkan pada gambar 4.28. Namun, dalam partisipasinya dalam aplikasi, operasi yang digunakan hanyalah ketiga operasi itu saja, sehingga class digambarkan demikian.

4.2.2.6 Class PlotDataTS

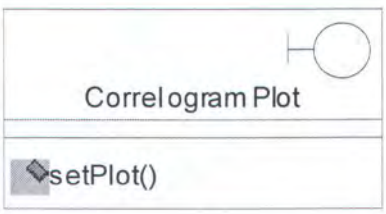
Class `PlotDataTS` digunakan untuk membuat antarmuka yang menampilkan grafik dari data runtut waktu yang dipilih. Gambar 4.29 menggambarkan class ini.



Gambar 4.29 Class PlotDataTS

4.2.2.7 Class CorrelogramPlot

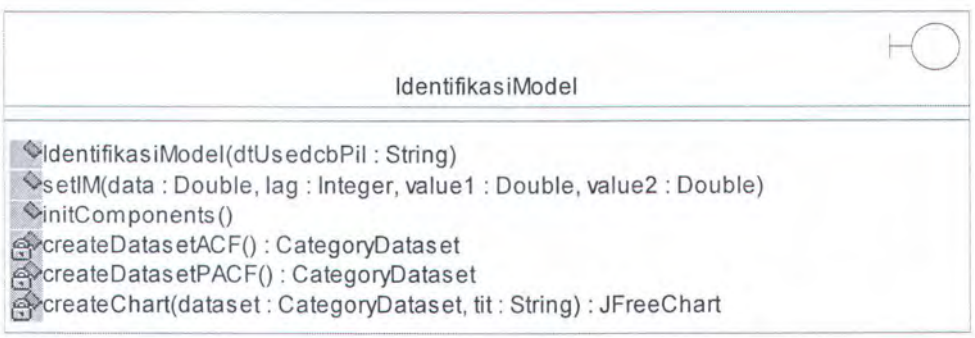
Class ini digunakan untuk membuat antarmuka yang menampilkan *correlogram* untuk ACF maupun PACF. Gambar 4.30 menggambarkan class `CorrelogramPlot`.



Gambar 4.30 Class CorrelogramPlot

4.2.2.8 Class IdentifikasiModel

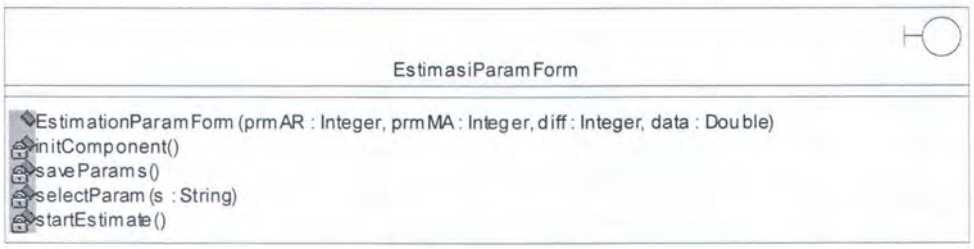
Class ini memberikan antarmuka bagi pengguna untuk menentukan model yang tepat berdasarkan *correlogram* ACF dan PACF data. Gambar 4.31 menggambarkan class ini.



Gambar 4.31 Class IdentifikasiModel

4.2.2.9 Class EstimasiParamForm

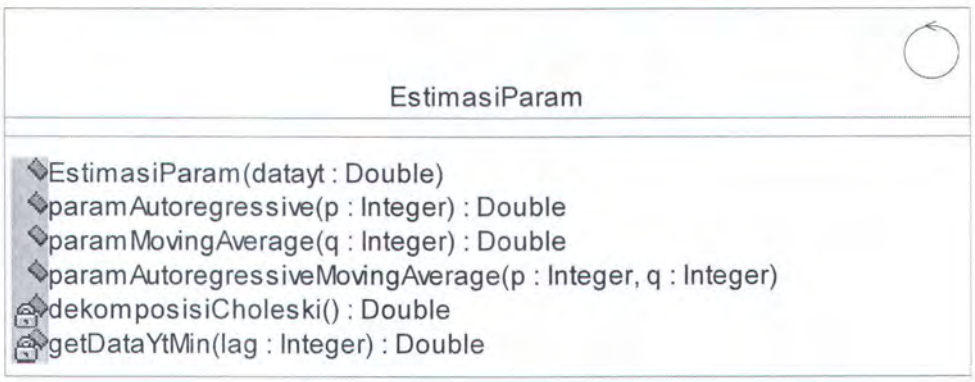
Class ini merupakan antarmuka yang diberikan untuk menampung catatan proses dan hasil penaksiran parameter untuk model ARIMA yang dipilih. Gambar 4.32 menggambarkan class ini.



Gambar 4.32 Class EstimasiParamForm

4.2.2.10 Class EstimasiParam

Class ini bertanggungjawab pada proses penaksiran parameter untuk model ARIMA yang dipilih, sehingga class ini akan dipanggil oleh class `EstimasiParamForm` untuk melakukan fungsinya. Gambar 4.33 menggambarkan class ini.



Gambar 4.33 Class EstimasiParam

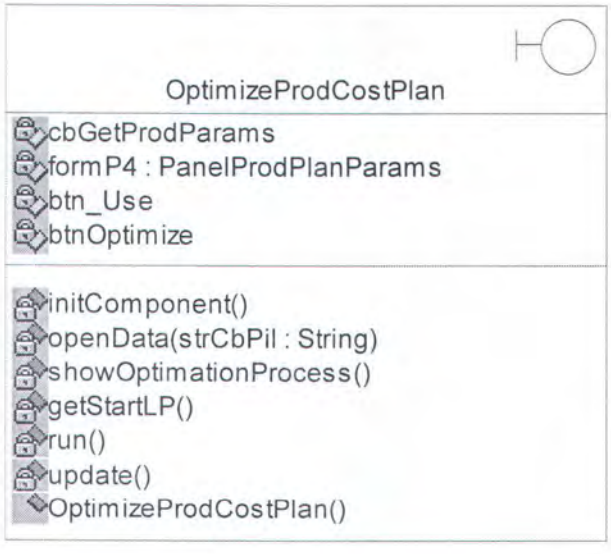
4.2.2.11 Class ForecastForm

Class ini merupakan antarmuka yang berfungsi sebagai tempat bagi pengguna untuk melakukan peramalan berdasarkan model ARIMA dan parameter yang telah dicari sebelumnya. Gambar 4.34 adalah gambaran dari class ini.

Gambar 4.34 Class ForecastForm

4.2.2.12 Class OptimizeProdCostPlan

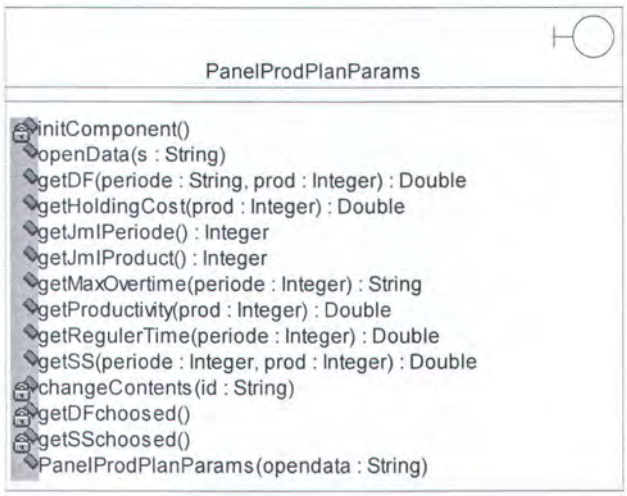
Class ini merupakan antarmuka yang menampung komponen-komponen untuk digunakan pengguna untuk mencari nilai biaya penyimpanan dan biaya lembur optimal. Gambar 4.35 menggambarkan class ini.



Gambar 4.35 Class OptimizeProdCostPlan

4.2.2.13 Class PanelProdPlanParams

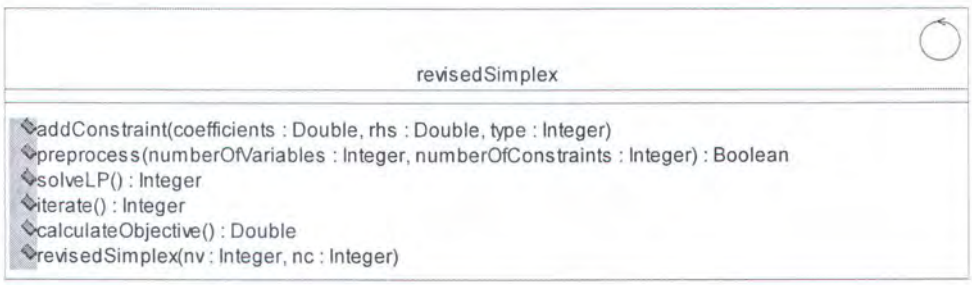
Class ini merupakan komponen panel yang dipanggil oleh class OptimizeProdCostPlan sebagai antarmuka yang menampung komponen-komponen untuk perhitungan nilai optimal biaya penyimpanan dan biaya lembur. Class ini digambarkan oleh gambar 4.36.



Gambar 4.36 Class PanelProdPlanParams

4.2.2.14 Class revisedSimplex

Class ini berfungsi untuk menghitung nilai optimal dari biaya penyimpanan dan biaya lembur berdasarkan pembatas-pembatasnya, dengan menggunakan pemrograman linier. Gambar 4.37 menggambarkan class ini.



Gambar 4.37 Class revisedSimplex

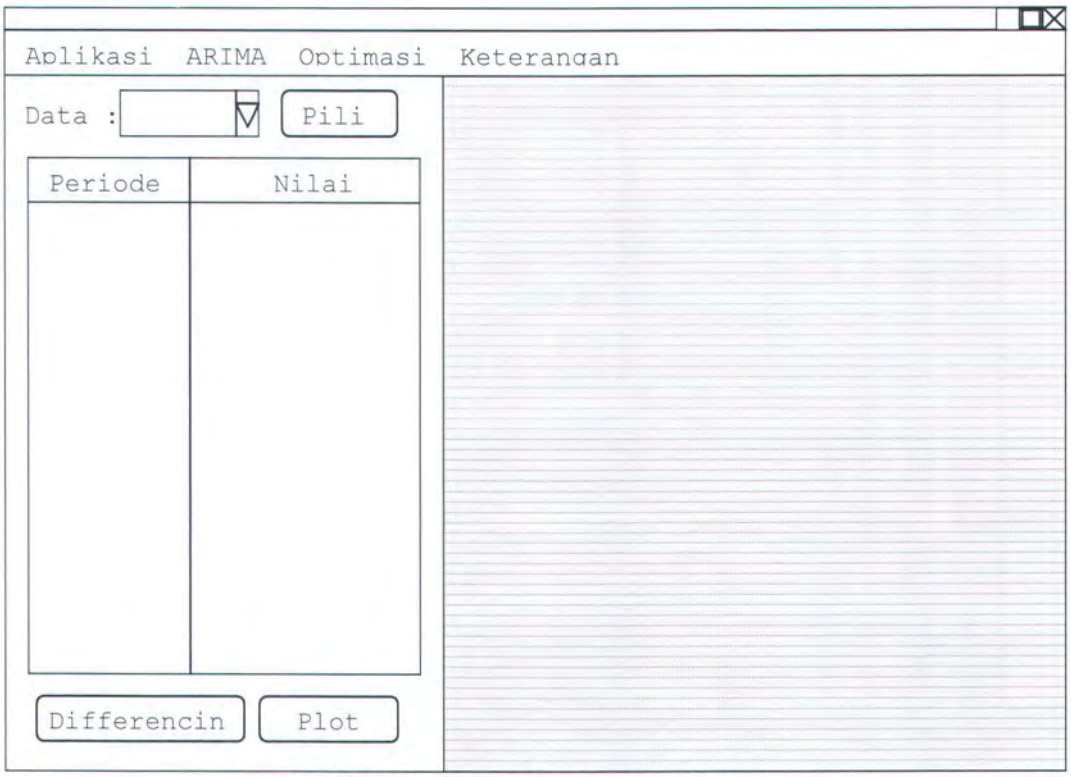
4.3 Desain Antarmuka

Antarmuka dari suatu aplikasi merupakan penghubung antara pengguna dengan aplikasi itu. Dengan penataan antarmuka yang baik, diharapkan pengguna tidak kesulitan dalam mengoperasikan aplikasi itu. Desain antarmuka dibutuhkan oleh pembuat aplikasi untuk menentukan komponen-komponen apa saja yang dibutuhkan dan bagaimana tata letak komponen-komponen itu, sebelum pembuatan aplikasi itu sendiri. Sub bab ini membahas mengenai desain antarmuka untuk aplikasi yang akan dibuat.

4.3.1 Antarmuka MainFrame

Seperti yang telah dituliskan pada sub bab sebelumnya, `MainFrame` merupakan antarmuka utama (frame utama) dari aplikasi, yang akan menghubungkan fitur-fitur aplikasi dengan pengguna. Gambar 4.38 merupakan desain antarmuka untuk `MainFrame`.

`MainFrame` memiliki menu bar yang menunya terdiri atas "Aplikasi", "ARIMA", "Optimasi", dan "Keterangan". Menu "Aplikasi" digunakan untuk mengatur aplikasi secara umum seperti koneksi database, ganti warna *dekstop*, dan keluar dari aplikasi. Menu "ARIMA" digunakan untuk melakukan peramalan dan analisis data sebelum peramalan. Menu "Optimasi" digunakan untuk melakukan optimasi biaya. Menu "Keterangan" digunakan hanya untuk informasi tambahan.



Gambar 4.38 Desain Antarmuka MainFrame

Pada bagian kiri frame terdapat `ComboBox` yang memuat daftar nama-nama data permintaan yang ada. Jika dipilih salah satu nama dari daftar tersebut, kemudian ditekan tombol "Pilih" disebelahnya, maka tabel dibawahnya akan berubah isinya menyesuaikan dengan nama data yang dipilih.

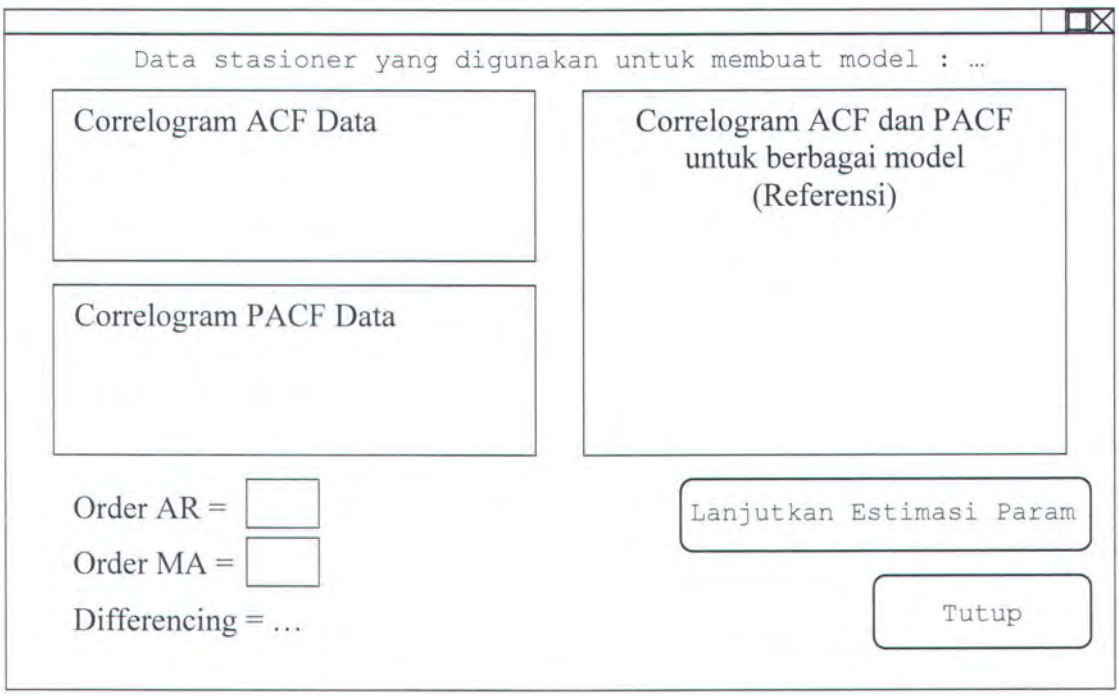
Di bawah tabel terdapat dua buah tombol, "Differencing" dan "Plot". Keduanya digunakan untuk memeriksa stasioneritas data.

4.3.2 Antarmuka Identifikasi Model

Seperti ditulis pada sub bab sebelumnya class `IdentifikasiModel` merupakan antarmuka yang akan digunakan pengguna untuk membandingkan *correlogram* ACF dan PACF data dengan *correlogram* teoritis untuk menentukan model ARIMA yang sesuai. Gambar 4.39 adalah desain antarmuka untuk class ini.

Pada bagian atas tertulis informasi data mana yang digunakan pengguna untuk menyusun model. Pada bagian kiri adalah *correlogram* data yang diobservasi, sedang bagian kanan adalah *correlogram* teoritis dari beberapa model

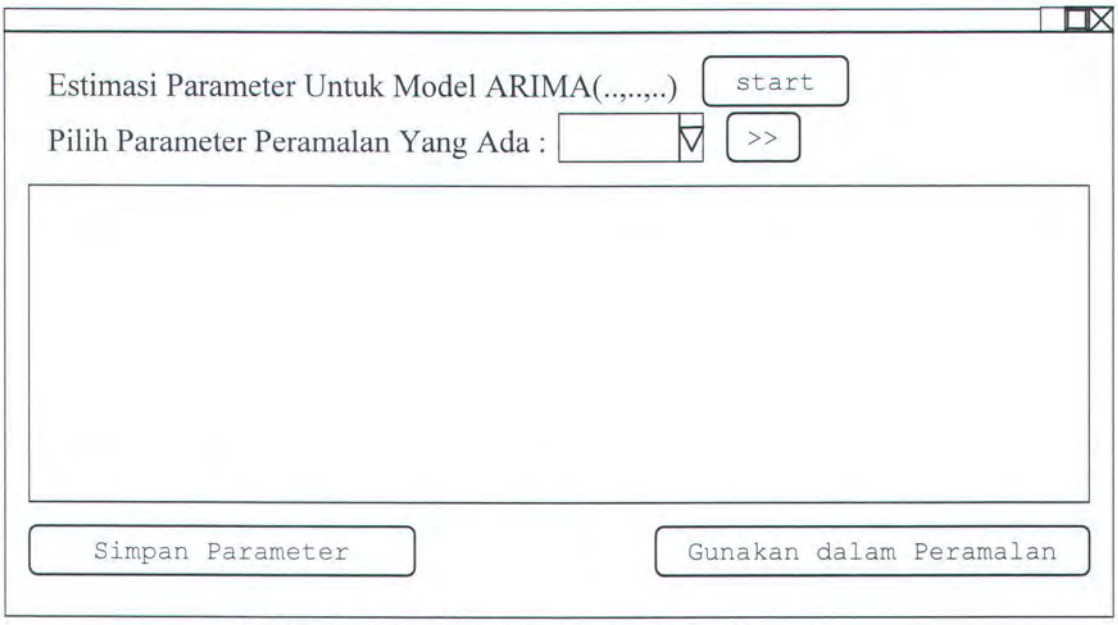
sebagai pembanding atau referensi. Pada bagian bawah, pengguna dapat memilih order AR (p) dan order MA (q) yang sesuai berdasarkan *correlogram* di atasnya. Setelah pengguna memilih order p dan q, pengguna dapat menekan tombol di pojok kanan bawah yaitu "Lanjutkan Estimasi Param" untuk melanjutkan ke penaksiran parameter.



Gambar 4.39 Desain Antarmuka IdentifikasiModel

4.3.3 Antarmuka Estimasi Parameter

Penaksiran parameter dilakukan setelah model ditentukan oleh pengguna, dan hasil dari penaksiran parameter adalah bilangan-bilangan sebagai konstanta dan koefisien dari model. Sehingga antarmuka yang dibuat hanya perlu menunjukkan perhitungan yang dilakukan aplikasi, karena yang diperlukan hanyalah hasil akhir penaksiran itu sendiri, yang akan digunakan untuk proses peramalan berikutnya. Desain antarmuka untuk penaksiran parameter ditunjukkan oleh gambar 4.40.

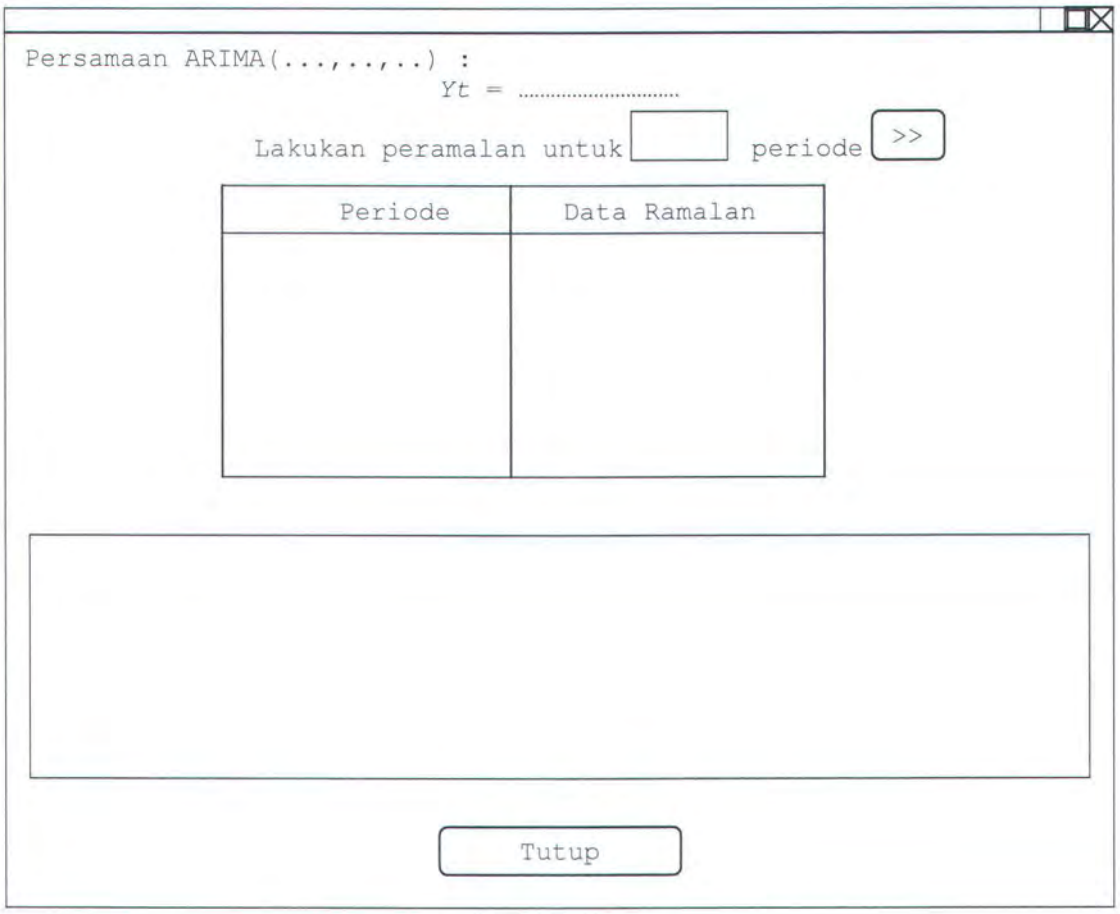


Gambar 4.40 Desain Antarmuka Estimasi Parameter

4.3.4 Antarmuka Peramalan Permintaan

Setelah model ditentukan, dan parameter untuk model telah berhasil ditaksir, pengguna langsung dapat melakukan peramalan menggunakan model dan parameter itu. Dalam melakukan peramalan, harus diketahui bagaimana modelnya, berapa periode yang akan diramal.

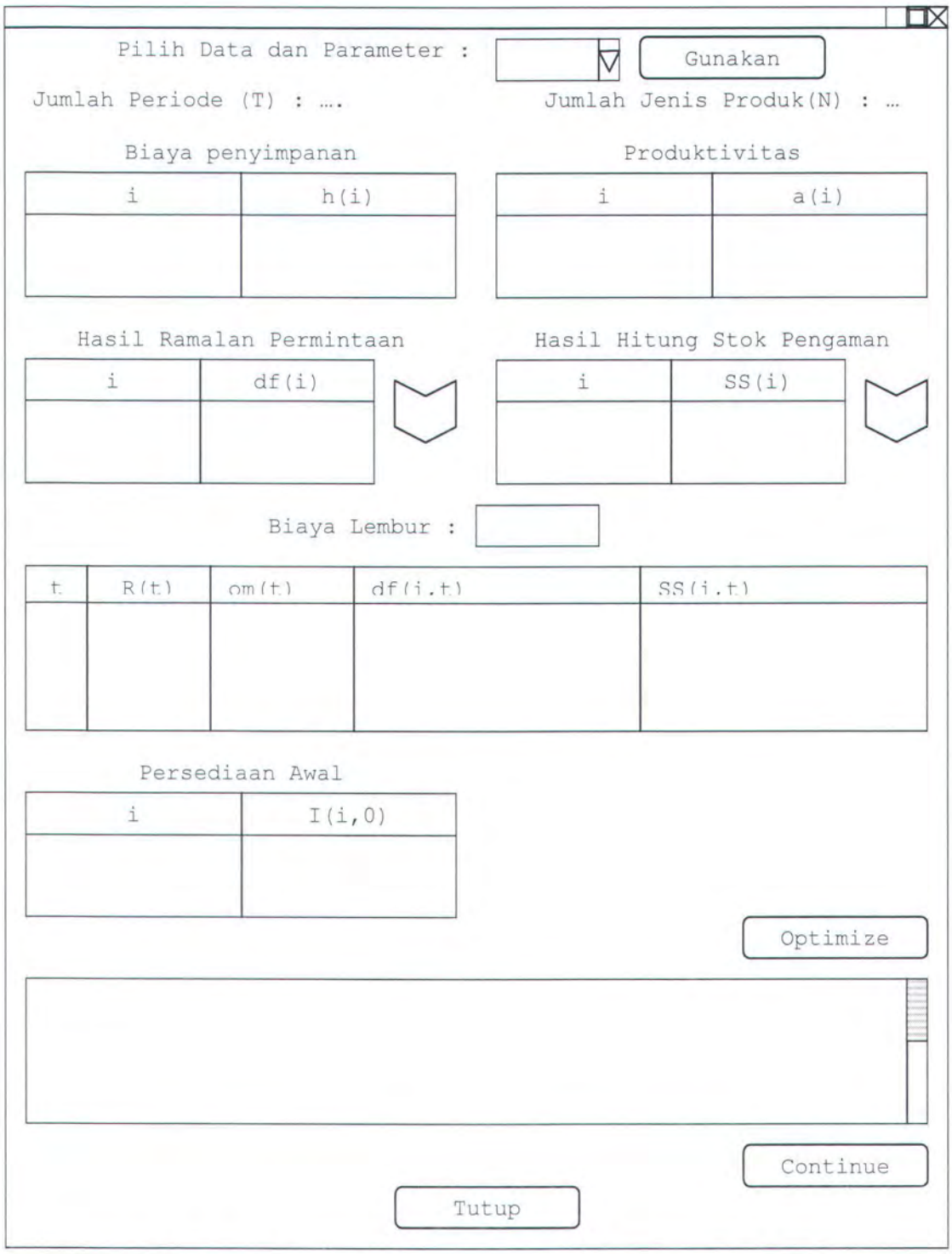
Selain itu juga akan ditambahkan tempat untuk perhitungan stok pengaman, yang nilainya dihitung berdasarkan perhitungan *error* estimasi paramete. Stok pengaman akan digunakan dalam menyusun perencanaan produksi. Gambar 4.41 merupakan desain antarmuka untuk peramalan permintaan.



Gambar 4.41 Desain Peramalan Permintaan

4.3.5 Antarmuka Perencanaan Biaya Penyimpanan dan Lembur

Sebelum membuat desain antarmuka untuk perencanaan biaya penyimpanan dan lembur perlu dilihat model perencanaan produksi yang telah dibahas pada sub bab 3.4, bahwa terdapat beberapa parameter dan model perencanaan produksi mengikuti model pemrograman linier. Desain antarmuka harus dapat memberikan ruang untuk setiap parameter tersebut serta memenuhi model pemrograman linier yang ada. Gambar 4.42 menunjukkan desain antarmuka untuk perencanaan biaya penyimpanan dan lembur ini.



Gambar 4.42 Desain Antarmuka Perencanaan Biaya Penyimpanan dan Lembur

BAB V

IMPLEMENTASI APLIKASI

BAB V

IMPLEMENTASI APLIKASI

Pada bab ini, dijelaskan mengenai implementasi dari hasil analisis dan desain yang telah dibahas pada bab IV. Pada bab IV telah diberikan analisis dan desain aplikasi yang menghasilkan kerangka sistem yang cukup spesifik. Implementasi aplikasi yang dilakukan meliputi penerapan hasil analisis dan desain aplikasi berkaitan dengan bahasa dan algoritma pemrograman serta penggunaan struktur data dalam membangun aplikasi. Secara keseluruhan, implementasi aplikasi yang dilakukan ditulis dalam bahasa pemrograman Java menggunakan kompiler j2sdk1.4.2_04.

5.1 Implementasi Use Case Memilih Data Runtut Waktu

Use case Memilih Data Runtut Waktu merupakan proses pemilihan data runtut waktu berupa data permintaan dari database yang akan dianalisis untuk membuat model ARIMA. Proses ini akan melibatkan beberapa class seperti yang terlihat pada gambar 4.7 pada bab sebelumnya. Dari gambar 4.7 tersebut tampak salah satu objek yang berpartisipasi adalah objek dari class `KoneksiDB`, dan method yang digunakan adalah konstruktor `KoneksiDB()` dan method `select()`.

Konstruktor `KoneksiDB` digunakan untuk mendefinisikan hubungan (koneksi) antara aplikasi dengan database. Karena database yang digunakan adalah MySQL, maka konstruktor hanya mendefinisikan koneksi untuk database MySQL dengan menggunakan *driver* untuk koneksi dengan database MySQL. Class *driver* untuk koneksi dengan database MySQL adalah `org.gjt.mm.mysql.Driver`. Segmen kode program 5.1 menunjukkan implementasi untuk konstruktor `KoneksiDB()`.

```
public KoneksiDB(String server,String db, String user,String password){
        try{
                Class.forName(jdbcDriver);

        conn=DriverManager.getConnection("jdbc:mysql://"+server+":3306/"+db,user,pa
ssword);
                stmt=conn.createStatement();
        }catch(ClassNotFoundException ex){
                ex.printStackTrace();
        }catch(SQLException se){
                se.printStackTrace();
        }
}
```

Segmen kode program 5. 1 Implementasi konstruktor KoneksiDB()

Konstruktor `KoneksiDB()` meminta 4 parameter bertipe `String` yaitu server, db, user, dan password. Parameter server adalah letak komputer server database yang digunakan, dapat berupa nama komputer (*host name*) atau nomor IP komputer. Parameter yang kedua, db, merupakan nama database yang terhubung dengan aplikasi. Parameter ketiga dan keempat berturut-turut merupakan nama *user* dan *password* untuk koneksi dengan database. Setelah koneksi dengan database dibangun, objek `stmt` dengan tipe `Statement` dibuat untuk mengirimkan kueri.

Konstruktor `KoneksiDB()` dipanggil oleh class `MainFrame` dalam method `initComponent()`. Konstruktor ini dipanggil pada saat method `initComponent()` melakukan *instanisasi* untuk atribut `superkonek`. Semua koneksi antara aplikasi dengan database dilakukan oleh atribut `superkonek` ini. Segmen kode program 5.2 adalah implementasi dari *instanisasi* atribut `superkonek`.

```
private void initComponents() {
        superkonek = new KoneksiDB("localhost","myts","root","");
        ...
        ...
}
```

Segmen kode program 5. 2 Instanisasi superkonek

Pada segmen kode program 5.2, parameter server diisi dengan "localhost", yang menunjukkan lokasi database berada pada komputer lokal. Begitu juga untuk parameter db, user, dan password yang secara berturut-turut berisi "myts","root","" menunjukkan nama database yang digunakan adalah myts, dengan *user* root dan *password* kosong (tanpa *password*).

Dalam database, data runtut waktu yang dapat dipilih oleh pengguna didaftarkan pada tabel `t_ts`. Nilai-nilai dari data tersebut untuk setiap periodenya berada pada tabel `t_penj`. Pada kondisi *default*, data yang terpilih adalah data runtut waktu dengan `id = 1`, sehingga nilai-nilai yang dimunculkan adalah nilai-nilai data pada tabel `t_penj` yang memiliki `idts = 1`. Nilai-nilai ini yang kemudian ditampilkan pada tabel di aplikasi. Potongan kode yang ditunjukkan oleh segmen kode program 5.3 merupakan implementasi untuk kondisi *default* tersebut.

Kondisi *default* secara otomatis terpilih pada saat aplikasi mulai dijalankan, untuk selanjutnya pengguna dapat memilih data runtut waktu yang akan diobservasinya. Pengguna dapat memilih data runtut waktu tersebut dari daftar yang ada di `ComboBox`, kemudian menekan tombol "Pilih". Segmen kode program 5.4 merupakan implementasi untuk penulisan daftar data runtut waktu ke `ComboBox`.

```
tabelMod.setColumnIdentifiers(new Object[]{"periode","nilai"});
try{
        rs=superkonek.select("t_penj.*","t_penj,t_ts","t_penj.idts=1 and
t_penj.idts=t_ts.id");
        if(rs!=null){
        while(rs.next()){
                tabelMod.addRow(new Object[]{rs.getString(3),rs.getString(4)});
        }
}
}catch(Exception exc){exc.printStackTrace();}
```

Segmen kode program 5. 3 Implementasi data runtut waktu pada kondisi default

```
try{
        rs=superkonek.select("*","t_ts",null);
        if(rs!=null){
                while(rs.next()){
                        cbTS.addItem(rs.getString(1)+" - "+rs.getString(2));
                }
        }
}catch(Exception exc){exc.printStackTrace();}
```

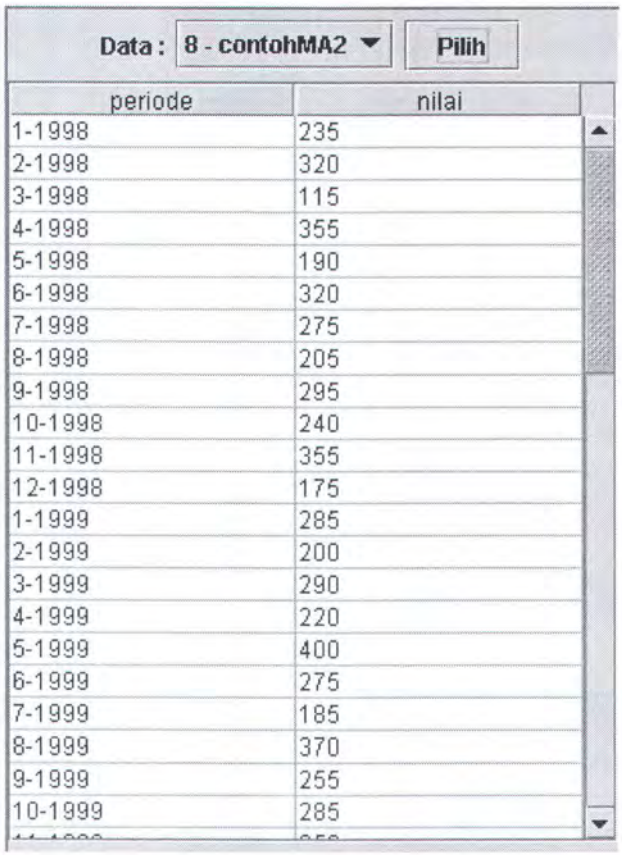
Segmen kode program 5. 4 Implementasi penulisan daftar data runtut waktu ke ComboBox

Sesaat setelah pengguna memilih data runtut waktu dari `ComboBox` dan menekan tombol "Pilih", aplikasi akan mengubah isi dari tabel yang berada di `MainFrame` sesuai dengan data runtut waktu yang dipilih pengguna. Class yang mengatur aksi perubahan ini adalah `FullAction` dengan method `actSetData()`. Implementasi dari aksi perubahan isi tabel di `MainFrame` ini ditunjukkan oleh segmen kode program 5.5.

```
public actSetData(){
        putValue(Action.NAME,"Set");
        putValue(Action.SHORT_DESCRIPTION,"Set Data");
        putValue(Action.MNEMONIC_KEY,new Integer('S'));
}
public void actionPerformed(ActionEvent event){
        try {
        int jumrow=MainFrame.tabelMod.getRowCount();
        for(int ii=0;ii<jumrow;ii++){
        if(MainFrame.tabelMod.getRowCount()>0){MainFrame.tabelMod.removeRow(0);}
        if(MainFrame.tabelMod.getColumnCount()>2){MainFrame.tabelMod.setColumnCount
(2);}
        }
        System.out.println(("")+MainFrame.cbTS.getSelectedItem()).charAt(1));
        ResultSet rs =
MainFrame.superkonek.select("t_penj.*","t_penj,t_ts","t_penj.idts='"+(""+MainFrame
.cbTS.getSelectedItem()).charAt(0)+"' and t_penj.idts=t_ts.id order by t_penj.id
asc");
        if(rs!=null){
                while(rs.next()){
                        MainFrame.tabelMod.addRow(new
Object[]{rs.getString(3),rs.getString(4)}); }
        }} catch (Exception e1) {e1.printStackTrace();}
}
```

Segmen kode program 5. 5 Implementasi perubahan isi tabel data runtut waktu

Secara keseluruhan, tampilan antarmuka dari aplikasi yang menunjukkan implementasi *use case Memilih Data Runtut Waktu* ditunjukkan oleh gambar 5.1. Letak komponen tersebut ada pada `MainFrame`, yaitu di sebelah kiri desktop aplikasi dan dapat disembunyikan apabila ternyata aplikasi membutuhkan ruang dekstop yang lebih luas.



Gambar 5.1 Tampilan antarmuka komponen aplikasi untuk memilih data runtut waktu

5.2 Implementasi Use Case Menganalisis Stasioneritas Data

Use case Menganalisis Stasioneritas Data merupakan serangkaian proses yang dilakukan untuk memeriksa kondisi stasioneritas data runtut waktu yang telah dipilih pengguna. Kondisi data runtut waktu yang stasioner diperlukan sebelum identifikasi model ARIMA dilakukan.

Pemeriksaan stasioneritas dapat dilakukan dengan melihat plot ACF, yaitu dengan memilih menu "ARIMA – Plot Autokorelasi – Plot ACF" yang terdapat pada `MainFrame`. Aksi yang dilakukan setelah pengguna memilih menu ini ditangani oleh class `FullAction` oleh method `actPlotACF()`. Implementasi dari aksi yang dilakukan oleh method `actPlotACF()` dituliskan pada segmen kode program 5.6.

```
public actPlotACF(){
        putValue(Action.NAME,"Plot ACF");
        putValue(Action.SHORT_DESCRIPTION,"Menampilkan nilai-nilai koefisien
autokorelasi");
        putValue(Action.MNEMONIC_KEY,new Integer('A'));
}
public void actionPerformed(ActionEvent event){
        JComboBox cbpil=new JComboBox();
        int jumkol=MainFrame.tabelMod.getColumnCount()-1;
        for(int k=0;k<jumkol;k++){
                cbpil.addItem(MainFrame.tabelMod.getColumnName(k+1));
        }
        JPanel comp=new JPanel();
        comp.add(new JLabel("Autokorelasi Data : "));
        comp.add(cbpil);
        int hsl=JOptionPane.showOptionDialog(null,comp,"Plot
Autokorelasi",JOptionPane.OK_CANCEL_OPTION,JOptionPane.INFORMATION_MESSAGE,null,nu
ll,"Hallo");
        if(hsl==0){
                double[] data=new double[MainFrame.tabelMod.getRowCount()];
                Object smt;
                TimeSeries ts=new TimeSeries("Data Uji Coba",Month.class);
                int[] lek=new int[MainFrame.tabelMod.getRowCount()];
                for(int y=0;y<MainFrame.tabelMod.getRowCount();y++){
                        lek[y]=y+1;
        smt=MainFrame.tabelMod.getValueAt(y,cbpil.getSelectedIndex()+1);
                        if(smt!=null){
                                System.gc();
                                data[y]=Double.parseDouble(""+smt);
                                ts.add(new Month(y+1, 1900), data[y]);
                        }
                }
                int leveldiff=0;
                if(cbpil.getSelectedIndex()>0){
leveldiff=Integer.parseInt((""+cbpil.getSelectedItem()).substring(5).trim());}
                ToolTS tul=new ToolTS(ts);
                double nilai[]=new double[lek.length];
                for(int y=0;y<lek.length-leveldiff;y++){
```

```
                    nilai[y]=tul.autoCorr(y+1);
                }
                System.gc();
                CorrelogramPlot coba=new CorrelogramPlot("Correlogram Autokorelasi
thd "+cbpil.getSelectedItem());
                coba.setPlot(lek,nilai);
                MainFrame.desktopPane.add(coba);
                try {
                        coba.pack();
                        coba.setVisible(true);
                        coba.setSelected(true);
                    } catch (Exception e1) {e1.printStackTrace();}
                }
        }
```

Segmen kode program 5. 6 Segmen kode program aksi plot ACF

Method `actPlotACF()` akan melakukan instanisasi objek dari class `ToolTS` untuk mendapatkan nilai ACF tiap lag. Method yang digunakan untuk mendapatkan nilai ACF adalah `autoCorr()`. Method ini melakukan perhitungan nilai koefisien autokorelasi dengan mengacu pada persamaan 2.1 di bab 2. Implementasi dari perhitungan tersebut dituliskan pada segmen kode program 5.7.

```
public double autoCorr(int lag){
        double sumdata=0,meandata=0;
        for(int k=0;k<TS.getItemCount();k++){
                data[k]=(TS.getValue(k)).doubleValue();
                sumdata=sumdata+data[k];
        }
        meandata=sumdata/TS.getItemCount();
        double pembilang=0,penyebut=0;
        for(int k=0;k<TS.getItemCount()-lag;k++){
                pembilang=pembilang+(data[k]-meandata)*(data[k+lag]-meandata);
        }
        for(int k=0;k<TS.getItemCount();k++){
                penyebut=penyebut+(Math.pow(data[k]-meandata,2));
        }
        return pembilang/penyebut;
}
```

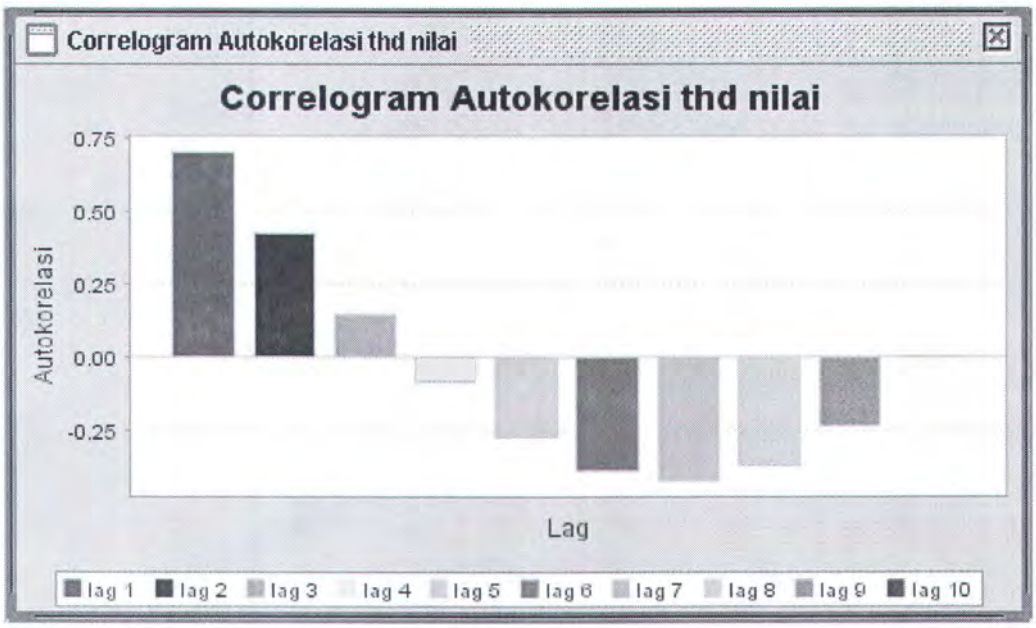
Segmen kode program 5. 7 Implementasi perhitungan ACF untuk setiap lag

Setelah nilai koefisien autokorelasi untuk setiap lag dihitung, maka nilai-nilai itu selanjutnya digambarkan pada *correlogram*. Aksi pembuatan

correlogram tersebut ditangani oleh class `CorrelogramPlot`. Segmen kode program 5.8 menunjukkan implementasi dari class `CorrelogramPlot`. Sedangkan gambar 5.2 merupakan contoh tampilan antarmuka *correlogram* dari ACF.

```
class CorrelogramPlot extends JInternalFrame{
        private int[] lags;
        private double[] values;
        public CorrelogramPlot(final String title){
                super(title,true,true,false,false);
        }
        public void setPlot(int[] lag,double[] value){
                lags=lag;
                values=value;
                final CategoryDataset dataset = createDataset();
                final JFreeChart chart = createChart(dataset);
                final ChartPanel chartPanel = new ChartPanel(chart);
                chartPanel.setPreferredSize(new Dimension(500, 270));
                setContentPane(chartPanel);
        }
        private CategoryDataset createDataset(){
                final DefaultCategoryDataset dataset = new
DefaultCategoryDataset();
                for(int i=0;i<lags.length;i++){
                        dataset.addValue(values[i],"lag "+lags[i],"");
                }
                return dataset;
        }
        private JFreeChart createChart(final CategoryDataset dataset) {
         final JFreeChart chart = ChartFactory.createBarChart(
             title,
             "Lag",
             "Autokorelasi",
             dataset,
             PlotOrientation.VERTICAL,
             true,
             true,
             false
         );
         return chart;
    }
}
```

Segmen kode program 5. 8 Implementasi class CorrelogramPlot



Gambar 5.2 Contoh tampilan antarmuka correlogram dari ACF

Selain dengan melihat plot ACF, stasioneritas data dapat dilihat langsung dari plot datanya. Untuk membuat plot data, pengguna dapat menekan tombol "Plot". Method `actPlotData()` pada class `FullAction` akan menangani aksi yang dilakukan setelah pengguna menekan tombol "Plot" tersebut. Implementasi dari method `actPlotData()` ditunjukkan pada segmen kode program 5.9.

```
public actPlotData(){
        putValue(Action.NAME,"Plot Data");
        putValue(Action.SHORT_DESCRIPTION,"Membuka Data");
        putValue(Action.MNEMONIC_KEY,new Integer('P'));
}
public void actionPerformed(ActionEvent event){
        int jumkol=MainFrame.tabelMod.getColumnCount()-1;
        TimeSeries[] ts=new TimeSeries[jumkol];
        if(jumkol>0){
                for(int kolke=0;kolke<MainFrame.tabelMod.getColumnCount()-
1;kolke++){
                        System.gc();
                        ts[kolke]=new
TimeSeries(""+MainFrame.tabelMod.getColumnName(kolke+1),Month.class);
                        for(int oi=0;oi<MainFrame.tabelMod.getRowCount();oi++){
                                System.gc();
                                StringTokenizer stoken=new
StringTokenizer(""+MainFrame.tabelMod.getValueAt(oi,0),"-");
```

```
                                int blntoken,thntoken;
                                blntoken=Integer.parseInt(stoken.nextToken());
                                thntoken=Integer.parseInt(stoken.nextToken());
                                if(MainFrame.tabelMod.getValueAt(oi,kolke+1)==null){
                                        ts[kolke].add(new Month(blntoken, thntoken),
null);
                                }
                                else{
                                        ts[kolke].add(new Month(blntoken, thntoken),
Double.parseDouble(""+MainFrame.tabelMod.getValueAt(oi,kolke+1)));
                                }
                        }
                }
        }
        Arr2DTS rrr=new Arr2DTS(ts);
        PlotDataTS dd=new PlotDataTS(rrr,"Plot Data");
        MainFrame.desktopPane.add(dd);
        try {
                dd.setSelected(true);
                dd.setSize(500,500);
                dd.setVisible(true);
        } catch (Exception e1) {e1.printStackTrace();}
}
```

Segmen kode program 5. 9 Implementasi method actPlotData()

Method `actPlotData()` selanjutnya akan memanggil class `PlotDataTS` untuk menggambar plot data. Segmen kode program 5.10 merupakan implementasi dari class `PlotDataTS`.

```
public class PlotDataTS extends JInternalFrame {
        private String judul;
        public PlotDataTS(Arr2DTS setdata,String title) {
                super(title,true,true,true,true);
                final XYDataset dataset = setdata.toDTS();
                final JFreeChart chart = createChart(dataset);
                final ChartPanel chartPanel = new ChartPanel(chart);
                judul=title;
                chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
                chartPanel.setMouseZoomable(true, false);
                setContentPane(chartPanel);
        }
        private JFreeChart createChart(final XYDataset dataset) {
                final JFreeChart chart = ChartFactory.createTimeSeriesChart(
                judul,
                "Periode", "Nilai",
```

```
                dataset,
                true,
                true,
                false
        );
        chart.setBackgroundPaint(Color.white);
        final StandardLegend sl = (StandardLegend) chart.getLegend();
        sl.setDisplaySeriesShapes(true);
        final XYPlot plot = chart.getXYPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
        plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
        plot.setDomainCrosshairVisible(true);
        plot.setRangeCrosshairVisible(true);
        final XYItemRenderer renderer = plot.getRenderer();
        if (renderer instanceof StandardXYItemRenderer) {
            final StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
            rr.setPlotShapes(true);
            rr.setShapesFilled(true);
            rr.setItemLabelsVisible(true);
        }
        final DateAxis axis = (DateAxis) plot.getDomainAxis();
        axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));
        return chart;
    }
}
```

Segmen kode program 5. 10 Implementasi class PlotDataTS

Apabila hasil dari pemeriksaan plot ACF maupun plot data, pengguna menyimpulkan bahwa data runtut waktu tersebut masih belum stasioner, maka pada data itu perlu dilakukan *differencing*. Pengguna dapat menekan tombol "Differencing" untuk melakukan *differencing* tingkat tertentu pada data runtut waktu bersangkutan. Method `actDifferencing()` pada class `FullAction` akan dijalankan sesaat setelah pengguna menekan tombol tersebut. Segmen kode program 5.11 merupakan implementasi dari method `actDifferencing()`.

```
public actDifferencing(){
        putValue(Action.NAME,"Differencing");
        putValue(Action.SHORT_DESCRIPTION,"Pembedaan");
        putValue(Action.MNEMONIC_KEY,new Integer('D'));
}
public void actionPerformed(ActionEvent event){
        int level=0;String slevel="";boolean ulang=true;
        while(ulang){
                slevel=JOptionPane.showInputDialog(null,"Tingkat differencing yang
dilakukan : ","Differencing Level",JOptionPane.OK_CANCEL_OPTION);
                try{
                        level=Integer.parseInt(slevel);
                        ulang=false;
                }catch(Exception e1){
                        if(slevel==null){ulang=false;}else{
        JOptionPane.showMessageDialog(null,"Masukkan angka!!");ulang=true;}}
        }
        if(slevel!=null||level>0){
                TimeSeries ts=new TimeSeries("Data Uji Coba",Month.class);
                for(int oi=0;oi<MainFrame.tabelMod.getRowCount();oi++){
                        ts.add(new Month(oi+1, 2001),
Double.parseDouble(""+MainFrame.tabelMod.getValueAt(oi,1)));
                }
                ToolTS pemb=new ToolTS(ts);
                double[] hslpemb=pemb.getDiff(level);
                MainFrame.tabelMod.addColumn("Diff "+level);
                for(int oi=level;oi<MainFrame.tabelMod.getRowCount();oi++){
                        MainFrame.tabelMod.setValueAt(""+hslpemb[oi-
level],oi,MainFrame.tabelMod.getColumnCount()-1);
                }
        }
}
```

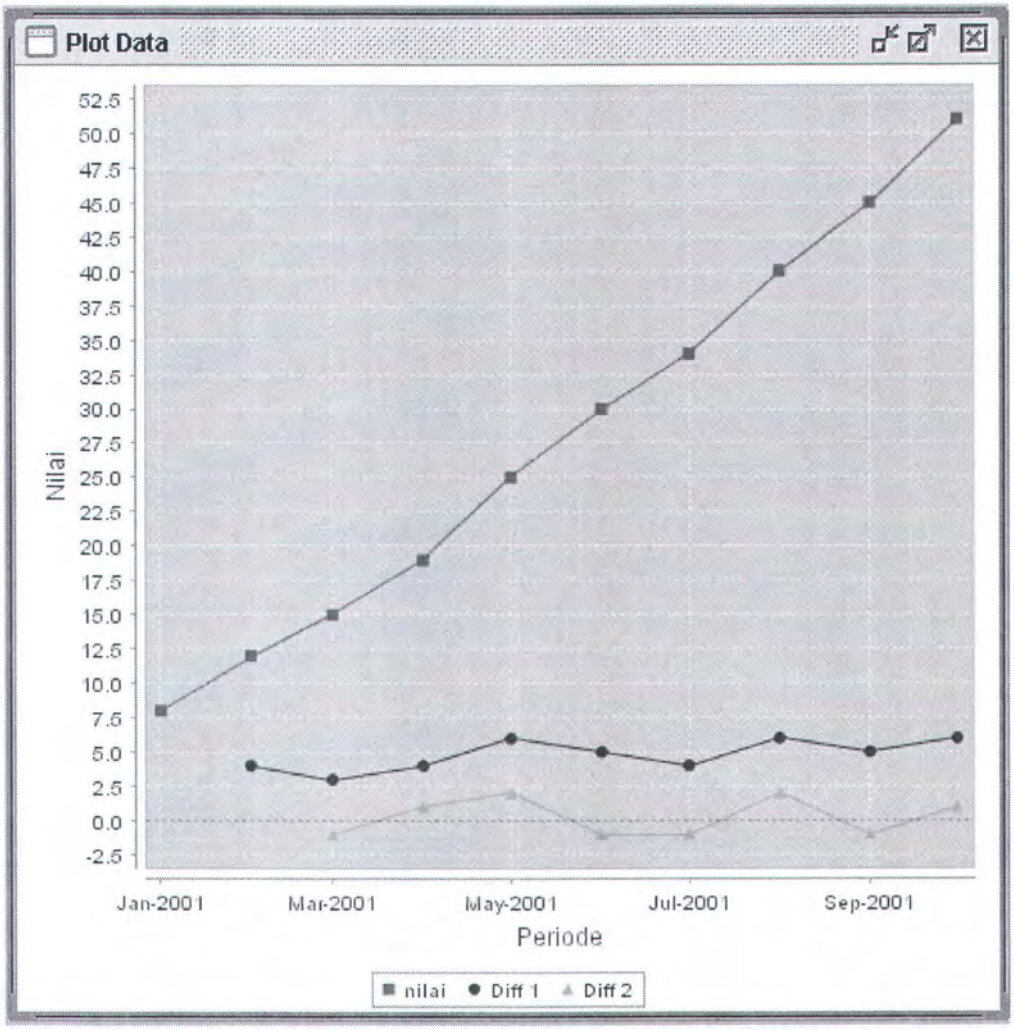
Segmen kode program 5. 11 Implementasi method actDifferencing()

Method `actDifferencing()` selanjutnya akan memanggil method `getDiff()` dari class `ToolTS` untuk mendapatkan serangkaian nilai-nilai hasil *differencing* tingkat tertentu dari data runtut waktu tersebut. Method `getDiff()` diimplementasikan pada segmen kode program 5.12.

```
public double[] getDiff(int level){
        for(int k=0;k<TS.getItemCount();k++){
                data[k]=(TS.getValue(k)).doubleValue();
        }
        double[] datadiff=new double[data.length];
        for(int poslev=0;poslev<level;poslev++){
                int i=0;
                if(poslev==0){
                        while(i<data.length-1){
                                datadiff[i]=data[i+1]-data[i];i++;
                        }
                }
                if(poslev>0){
                        while(i<data.length-poslev-1){
                                datadiff[i]=datadiff[i+1]-datadiff[i];i++;
                        }
                }
        }return datadiff;
}
```

Segmen kode program 5. 12 Implementasi method getDiff()

Nilai-nilai dari data hasil proses *differencing* ditampilkan pada tabel di `MainFrame` dengan menambah kolom "Diff n" disebelah kanan, dimana n merupakan tingkat *differencing*. Nilai-nilai hasil *differencing* juga dapat digambarkan dengan cara yang sama dengan plot data dan akan ditampilkan bersama-sama dengan plot data asli. Contoh tampilan antarmuka plot data asli dan hasil *differencing* digambarkan pada gambar 5.3.



Gambar 5.3 Contoh plot data asli dan hasil differencing tingkat 1 dan 2

5.3 Implementasi Use Case Menjalankan Peramalan Permintaan

Use case Menjalankan Peramalan Permintaan merupakan proses peramalan permintaan menggunakan data runtut waktu yang telah stasioner. Untuk melakukan proses peramalan, pengguna dapat memilih menu "ARIMA – Proses Peramalan" pada `MainFrame`. Aksi yang akan dijalankan oleh aplikasi sesaat setelah pengguna memilih menu tersebut ditangani oleh class `FullAction` dengan method `actForecast()`. Method ini akan memanggil objek `JOptionPane` yang akan meminta masukan berupa pilihan penggunaan data stasioner dan

jumlah record data untuk observasi. Segmen kode program 5.13 menunjukkan implementasi dari method ini.

```
public actForecast(){
        putValue(Action.NAME,"Proses Peramalan");
        putValue(Action.SHORT_DESCRIPTION,"Proses Peramalan dengan ARIMA");
        putValue(Action.MNEMONIC_KEY,new Integer('I'));
}
public void actionPerformed(ActionEvent event){
        JComboBox cbpil=new JComboBox();
        JTextField tfJumDatMod=new JTextField("0",5);
        int jumkol=MainFrame.tabelMod.getColumnCount()-1;
        for(int k=0;k<jumkol;k++){
                cbpil.addItem(MainFrame.tabelMod.getColumnName(k+1));
        }
        JPanel comp=new JPanel(new GridLayout(2,1));
        JPanel penelPilihDataSta=new JPanel();
        penelPilihDataSta.add(new JLabel("Pilih Data Yang Telah Stasioner : "));
        penelPilihDataSta.add(cbpil);
        JPanel penelJumlahDataModel=new JPanel();
        penelJumlahDataModel.add(new JLabel("Jumlah record yang digunakan "));
        penelJumlahDataModel.add(tfJumDatMod);
        penelJumlahDataModel.add(new JLabel("record pertama"));
        comp.add(penelPilihDataSta);
        comp.add(penelJumlahDataModel);
        int jumBwtModel=0;
        int hsl=2,salah=0;
        do{
        if(salah==1){JOptionPane.showMessageDialog(null,"Masukkan Bilangan!!");}
        salah=0;
        try{
                hsl=JOptionPane.showOptionDialog(null,comp,"Pilih Data
Stasioner",JOptionPane.OK_CANCEL_OPTION,JOptionPane.INFORMATION_MESSAGE,null,null,
"");
                System.out.println(hsl);
                jumBwtModel=Integer.parseInt(tfJumDatMod.getText());
        }catch(Exception ee){salah=1; }
        }while(!(hsl==2||(salah==0&&jumBwtModel>0&&jumBwtModel<=MainFrame.tabelMod.
getRowCount())));
        if(hsl==0){
                if(jumBwtModel<MainFrame.tabelMod.getRowCount()){
                JPanel compl=new JPanel(new GridLayout(2,1));
                JPanel penelTfSaveAs=new JPanel();
                JTextField tfSaveAs=new
JTextField("real_"+cbpil.getSelectedItem()+"_"+(jumBwtModel+1)+"_"+(MainFrame.tabe
lMod.getRowCount()-jumBwtModel),15);
                penelTfSaveAs.add(new JLabel("Nama :"));
```

```
                penelTfSaveAs.add(tfSaveAs);
                compl.add(new JLabel("Simpan sisanya
("+(MainFrame.tabelMod.getRowCount()-jumBwtModel)+" record) sebagai data real
permintaan?"));
                compl.add(penelTfSaveAs);
                int setRealD=JOptionPane.showOptionDialog(null,compl,"Simpan Sisa
Record",JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE,null,null,"");
                if(setRealD==0){
                try{
                int ins =
MainFrame.superkonek.insert("t_realdemand","'','"+tfSaveAs.getText()+"','t_ts',"+(
jumBwtModel+1)+","+(MainFrame.tabelMod.getRowCount()-jumBwtModel)+",'diambil dari
record sisa data identifikasi model'");
                String newestidrealD="";
                ResultSet rs =
MainFrame.superkonek.select("max(id)","t_realdemand",null);
                if(rs!=null){
                        if(rs.next()){
                                newestidrealD=rs.getString(1);
                        }
                }
                for(int k=jumBwtModel;k<MainFrame.tabelMod.getRowCount();k++){
                        ins=MainFrame.superkonek.insert("t_realdemanddata",
        "'','"+newestidrealD+"','"+MainFrame.tabelMod.getValueAt(k,0)+"','"+MainFra
me.tabelMod.getValueAt(k,1)+"'");
                }
                }catch(Exception ee){ee.printStackTrace();}
                }}
        double[] data=new double[jumBwtModel];
        Object smt;
        TimeSeries ts=new TimeSeries("Data Uji Coba",Month.class);
        int[] lek=new int[jumBwtModel];
        for(int y=0;y<jumBwtModel;y++){
                lek[y]=y+1;
                smt=MainFrame.tabelMod.getValueAt(y,cbpil.getSelectedIndex()+1);
                if(smt!=null){
                        System.gc();
                        data[y]=Double.parseDouble(""+smt);
                        ts.add(new Month(y+1, 1900), data[y]);
                }
        }
        int leveldiff=0;
        if(cbpil.getSelectedIndex()>0){leveldiff=Integer.parseInt((""+cbpil.getSele
ctedItem()).substring(5).trim());}
        ToolTS tul=new ToolTS(ts);
        double acf[]=new double[lek.length];
        double pacf[]=new double[lek.length];
```

```
        for(int y=0;y<lek.length-leveldiff;y++){
                acf[y]=tul.autoCorr(y+1);
                pacf[y]=tul.parsAutoCorr(y+1);
        }
        System.gc();
        IdentifikasiModel coba=new IdentifikasiModel(""+cbpil.getSelectedItem());
        coba.setIM(data,lek,acf,pacf);
        MainFrame.desktopPane.add(coba);
        try {
                coba.setSize(700,600);
                coba.setVisible(true);
                coba.setSelected(true);
        } catch (Exception e1) {e1.printStackTrace();}
        }
}
```

Segmen kode program 5. 13 Implementasi method actForecast()

Setelah pengguna memasukkan pilihan data stasioner dan jumlah record data yang digunakan, method `actForecast()` akan memanggil objek dari class `ToolTS`. Objek ini akan menggunakan method `autoCorr()` dan `parsAutoCorr()` untuk mendapatkan nilai ACF dan PACF dari sejumlah record pilihan data stasioner tadi. Kemudian method `actForecast()` akan melakukan *instanisasi* objek dari class `IdentifikasiModel`. Objek ini kemudian memanggil method `setIM()` dengan mengambil nilai parameter ketiga dan keempatnya berupa nilai ACF dan PACF yang telah didapatkan sebelumnya. Method `setIM()` selanjutnya akan menginisialisasi komponen-komponen untuk class `IdentifikasiModel` dengan method `initComponents()`. Implementasi method `autoCorr()` telah diberikan pada segmen kode program 5.7, sedangkan untuk method `parsAutoCorr()` yang mengacu pada persamaan 2.2 dan 2.3 di bab 2, ditunjukkan oleh segmen kode program 5.14. Dan implementasi untuk method `setIM()` ditunjukkan oleh segmen kode program 5.15.

```
public double parsAutoCorr(int ka){
        double[] R=new double[ka];
        double[][] r=new double[ka][ka];
        for(int i=0;i<ka;i++){
                R[i]=autoCorr(i+1);
        }
        r[0][0]=R[0];  // r11 = r1
        if(ka>1){
                for(int k=1;k<ka;k++){
                        for(int j=k;j>=0;j--){
                                if(j==k){
                                        double sigmaAtas=0,sigmaBawah=0;
                                        for(int je=0;je<j;je++){
                                                int kha=k+1,jhe=je+1;
                                                sigmaAtas=sigmaAtas+(r[k-
1][je]*R[kha-jhe-1]);
                                                sigmaBawah=sigmaBawah+(r[k-
1][je]*R[je]);
                                        }
                                        r[k][j]=(R[k]-sigmaAtas)/(1-sigmaBawah);
                                }
                                else if(j<k){
                                        r[k][j]=r[k-1][j]-r[k][k]*r[k-1][k-j-1];
                                }
                        }
                }
        }
        return r[ka-1][ka-1];
}
```

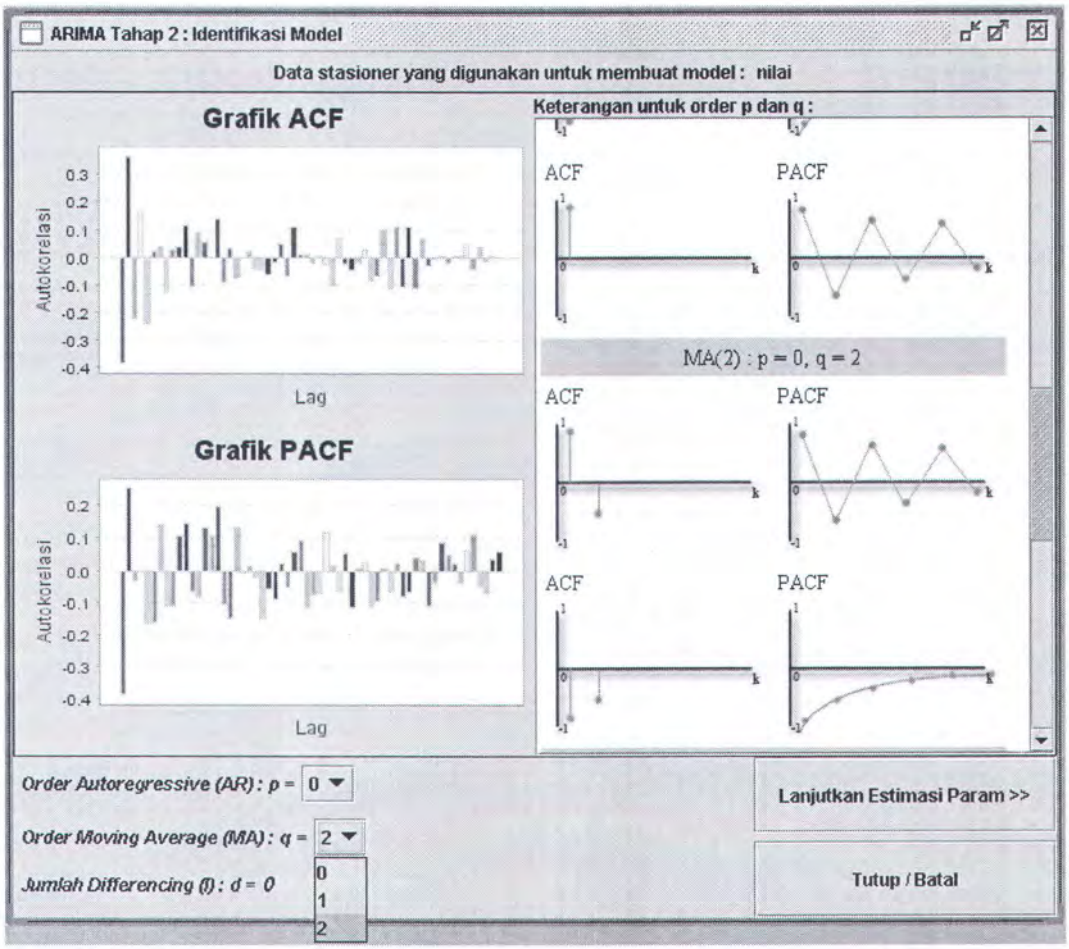
Segmen kode program 5. 14 Implementasi method parsAutoCorr()

```
public void setIM(double[] data,int[] lag,double[] value1,double[] value2){
        dataY=data;lags=lag;
        acf=value1;
        pacf=value2;
        initComponents();
}
```

Segmen kode program 5. 15 Implementasi method setIM()

Class `IdentifikasiModel` digunakan untuk melakukan identifikasi model terhadap data runtut waktu disesuaikan dengan karakteristik model ARIMA teoritis. Identifikasi model dilakukan dengan membandingkan plot ACF dan plot PACF data runtut waktu bersangkutan dengan kombinasi plot ACF dan plot

PACF model ARIMA teoritis. Tampilan antarmuka dari class `IdentifikasiModel` digambarkan pada gambar 5.4.



Gambar 5.4 Tampilan antarmuka class IdentifikasiModel

Setelah pengguna menemukan model ARIMA yang paling sesuai dengan data runtut waktu, langkah selanjutnya adalah penaksiran (estimasi) parameter. Untuk melanjutkan ke tahap penaksiran parameter, pengguna melakukannya dengan menekan tombol "Lanjutkan Estimasi Param". Aksi yang dilakukan sesaat setelah tombol ini ditekan adalah melakukan *instanisasi* objek dari class `EstimasiParamForm` dengan membawa beberapa nilai parameter. Segmen kode program 5.16 menunjukkan aksi dari penekanan tombol "Lanjutkan Estimasi Param".

```
btnLanjut.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent event){
        dispose();
        int jumAR=Integer.parseInt(""+cbAR.getSelectedItem());
        int jumMA=Integer.parseInt(""+cbMA.getSelectedItem());
        EstimasiParamForm coba=new EstimasiParamForm(jumAR,jumMA,levelDiff,dataY);
        MainFrame.desktopPane.add(coba);
        try {
                coba.setSize(500,350);
                coba.setVisible(true);
                coba.setSelected(true);
        } catch (Exception e1) {e1.printStackTrace();}
}});
```

Segmen kode program 5. 16 Implementasi aksi untuk tombol "Lanjutkan Estimasi Param"

Class `EstimasiParamForm` menyediakan dua pilihan untuk menentukan parameter yang akan digunakan. Pilihan pertama yaitu pengguna menekan tombol "start", sehingga aplikasi akan melakukan penaksiran parameter dengan beberapa perhitungan dan iterasi hingga nilai parameter yang mendekati cocok ditemukan. Pilihan kedua adalah pengguna memilih nilai-nilai parameter yang telah tersimpan sebelumnya dari daftar parameter yang tersedia di `ComboBox`.

Apabila pengguna memilih pilihan pertama, maka aplikasi melalui class `EstimasiParamForm` akan menjalankan method `startEstimate()`. Method `startEstimate()` akan mencari nilai parameter yang sesuai dengan model yang dipilih sebelumnya. Implementasi dari method `startEstimate()` dituliskan pada segmen kode program 5.17.

Method `startEstimate()` ini akan melakukan *instanisasi* objek dari class `EstimasiParam`. Objek inilah yang bertanggungjawab dalam menemukan nilai parameter yang sesuai untuk model.

```
private void startEstimate(){
        EstimasiParam EP=new EstimasiParam(dataY);
        if(p==1&&q==0){
                double[] pp=new double[2];
                pp=EP.paramAutoregressive(1);
                textArea.append("param AR (a) = "+pp[0]+" \n");
                textArea.append("param AR (b1) = "+pp[1]);
                paramToSave=pp[0]+"|"+pp[1];
        }else if(p==2&&q==0){
                double[] pp=new double[3];
                pp=EP.paramAutoregressive(2);
                textArea.append("param AR (a) = "+pp[0]+" \n");
                textArea.append("param AR (b1) = "+pp[1]+" \n");
                textArea.append("param AR (b2) = "+pp[2]);
                paramToSave=pp[0]+"|"+pp[1]+"|"+pp[2];
        }else if(p==0&&q==1){
                double[] pp=new double[2];
                pp=EP.paramMovingAverage(1);
                textArea.append("param MA (c) = "+pp[0]+" \n");
                textArea.append("param MA (m1) = "+pp[1]);
                paramToSave=pp[0]+"|"+pp[1];
        }else if(p==0&&q==2){
                double[] pp=new double[3];
                pp=EP.paramMovingAverage(2);
                textArea.append("param MA (c) = "+pp[0]+" \n");
                textArea.append("param MA (m1) = "+pp[1]+" \n");
                textArea.append("param MA (m2) = "+pp[2]);
                paramToSave=pp[0]+"|"+pp[1]+"|"+pp[2];
        }
        paramUsed=paramToSave;
    }
```

Segmen kode program 5. 17 Implementasi method startEstimate()

Jika model yang dipilih adalah model *autoregressive*, maka yang dijalankan adalah method `paramAutoregressive()`. Jika model yang dipilih adalah model *moving average*, maka yang dijalankan adalah method `paramMovingAverage()`. Jika model yang dipilih adalah model campuran *autoregressive* dan *moving average*, maka yang dijalankan adalah method `paramAutoregressiveMovingAverage()`. Masing-masing method tersebut membawa parameter berupa order p, q, atau p dan q. Implementasi dari method-

method tersebut berturut-turut ditunjukkan pada segmen program 5.18, 5.19, dan 5.20.

```
public double[] paramAutoregressive(int p){
        double[] returnval=new double[p+1];
        double[] dataYtMin=new double[dataYt.length];
        dataYtMin=getDataYtMin(p);
        if(p==1){
                double BSum1=0,BSum2=0,BSum3=0,BSum4=0;
                for(int t=0;t<dataYt.length;t++){
                        BSum1=BSum1+(dataYtMin[t]*dataYt[t]);
                        BSum2=BSum2+dataYtMin[t];
                        BSum3=BSum3+dataYt[t];
                        BSum4=BSum4+Math.pow(dataYtMin[t],2.0);
                }
                returnval[1]=(dataYt.length*BSum1-
BSum2*BSum3)/(dataYt.length*BSum4-(Math.pow(BSum2,2.0)));
                returnval[0]=(BSum3/dataYt.length)-
returnval[1]*(BSum2/dataYt.length);
                EstimasiParamForm.textArea.append("Regresi Linier Sederhana \n");
        }
        else if(p==2){
                EstimasiParamForm.textArea.append("Dekomposisi Choleski \n");
                returnval=dekomposisiCholeski();
        }
        return returnval;
}
```

Segmen kode program 5. 18 Implementasi method paramAutoregressive()

```
public double[] paramMovingAverage(int q){
        double[] returnval=new double[q+1];
        if(q==1){
                double sumdata=0.0;
                for(int t=0;t<dataYt.length;t++){
                        sumdata=sumdata+dataYt[t];
                }
                double c=sumdata/dataYt.length;
                double batasAtas=1,batasBawah=-1,btsAtas=1,btsBwh=-1;
                double m1=batasAtas;
                double minSSR=0.0,M1=0.0,Zt,et;
                int iterke=0;
                double step=0.05;
                //START PENCARIAN DIKEDALAMAN I
                do{
```

```
                        double SSR=0.0;
                        double et_1=0.0;//dianggap 0
                        for(int t=0;t<dataYt.length;t++){
                                Zt=c-m1*et_1;
                                et=dataYt[t]-Zt;//dataYt[t]
                                SSR=SSR+Math.pow(et,2);
                                et_1=et;
                        }
                        if(iterke==0){
                                minSSR=SSR;
                                M1=m1;
                        }
                        if(SSR<minSSR){
                                minSSR=SSR;
                                M1=m1;
                                btsAtas=m1+step;
                                btsBwh=m1-step;
                        }
                        EstimasiParamForm.textArea.append("iterasi "+iterke+" :
m1="+m1+"\t SSR="+SSR+" \n");
                        m1=m1-step;
                        iterke++;
                }while(m1>batasBawah&&m1<batasAtas);
                //START PENCARIAN DIKEDALAMAN II...VI
                for(int j=0;j<5;j++){
                        batasBawah=btsBwh;
                        batasAtas=btsAtas;
                        m1=batasAtas;
                        step=step/20;
                        do{
                                double SSR=0.0;
                                double et_1=0.0;//dianggap 0
                                for(int t=0;t<dataYt.length;t++){//dataYt
                                        Zt=c-m1*et_1;
                                        et=dataYt[t]-Zt;//dataYt[t]
                                        SSR=SSR+Math.pow(et,2);
                                        et_1=et;
                                }
                                if(SSR<minSSR){
                                        minSSR=SSR;
                                        M1=m1;
                                        btsAtas=m1+step;
                                        btsBwh=m1-step;
                                }
                                EstimasiParamForm.textArea.append("iterasi
"+iterke+" : m1="+m1+"\t SSR="+SSR+" \n");
                                m1=m1-step;
```

```
                        iterke++;
                    }while(m1>batasBawah&&m1<batasAtas);
                }//STOP PENCARIAN DIKEDALAMAN II...VI
                EstimasiParamForm.textArea.append("\n M1="+M1+"\t minSSR="+minSSR+"
\n");
                returnval[0]=c;
                returnval[1]=M1;
        }
        else if(q==2){
                double sumdata=0.0;
                for(int t=0;t<dataYt.length;t++){//dataYt
                        sumdata=sumdata+dataYt[t];//dataYt
                }
                double c=sumdata/dataYt.length;//dataYt
                double batasAtasm2=1,batasBawahm2=-1,btsAtasm2=1,btsBwhm2=-1;
                double m2=batasAtasm2;
                double minSSR=0.0,M1=0.0,M2=1,Zt,et;
                int iterke=0;
                double stepm1=0.1;double stepm2=0.1;//UBAH untuk ketelitian
                double batasAtasm1=1-m2,batasBawahm1=m2-1,btsAtasm1=1-
m2,btsBwhm1=m2-1;
                double m1=batasAtasm1;
                //START PENCARIAN DIKEDALAMAN I
                double SSR;
                double et_1;
                double et_2;
                do{
                        do{
                                SSR=0.0;
                                et_1=0.0;
                                et_2=0.0;
                                for(int t=0;t<dataYt.length;t++){//dataYt
                                        Zt=c-m1*et_1-m2*et_2;
                                        et=dataYt[t]-Zt;//dataYt[t]
                                        SSR=SSR+Math.pow(et,2);
                                        et_2=et_1;
                                        et_1=et;
                                }
                                if(iterke==0){
                                        minSSR=SSR;
                                        M1=m1;
                                        M2=m2;
                                }
                                if(SSR<minSSR){
                                        minSSR=SSR;
                                        M1=m1;
                                        M2=m2;
```

```
                                btsAtasm1=m1+stepm1;
                                btsBwhm1=m1-stepm1;
                                btsAtasm2=m2+stepm2;
                                btsBwhm2=m2-stepm2;
                        }
                        EstimasiParamForm.textArea.append("iterasi
"+iterke+" : m1="+m1+" ## m2="+m2+"\t SSR="+SSR+" \n");
                        m1=m1-stepm1; //m1=-0.05
                        iterke++;
                }while(m1>batasBawahm1&&m1<batasAtasm1);
                m2=m2-stepm2;//m2=0.95
                batasBawahm1=batasBawahm1-stepm2;
                batasAtasm1=batasAtasm1+stepm2;
                m1=batasAtasm1;
        }while(m2>batasBawahm2&&m2<batasAtasm2);
        //START PENCARIAN DIKEDALAMAN II...VI
        for(int j=0;j<5;j++){
                stepm1=stepm1/100;
                stepm2=stepm2/100;
                batasBawahm2=btsBwhm2;
                batasAtasm2=btsAtasm2;
                m2=batasAtasm2;
                do{
                        batasBawahm1=btsBwhm1;
                        batasAtasm1=btsAtasm1;
                        m1=batasAtasm1;
                        do{
                                SSR=0.0;
                                et_1=0.0;
                                et_2=0.0;
                                for(int t=0;t<dataYt.length;t++){//dataYt
                                        Zt=c-m1*et_1-m2*et_2;
                                        et=dataYt[t]-Zt;//dataYt[t]
                                        SSR=SSR+Math.pow(et,2);
                                        et_2=et_1;
                                        et_1=et;
                                }
                                if(SSR<minSSR){
                                        minSSR=SSR;
                                        M1=m1;
                                        M2=m2;
                                        btsAtasm1=m1+stepm1;
                                        btsBwhm1=m1-stepm1;
                                        btsAtasm2=m2+stepm2;
                                        btsBwhm2=m2-stepm2;
                                }
```

```
                                        EstimasiParamForm.textArea.append("iterasi
"+iterke+" : m1="+m1+" ## m2="+m2+"\t SSR="+SSR+" \n");
                                        m1=m1-stepm1;
                                        iterke++;
                                    }while(m1>batasBawahm1&&m1<batasAtasm1);
                                    m2=m2-stepm2;
                                    batasBawahm1=batasBawahm1-stepm2;
                                    batasAtasm1=batasAtasm1+stepm2;
                                    m1=batasAtasm1;
                                }while(m2>batasBawahm2&&m2<batasAtasm2);
            }/**/
            returnval[0]=c;
            returnval[1]=M1;
            returnval[2]=M2;
            EstimasiParamForm.textArea.append("\n M1="+M1+" M2="+M2+"\t
minSSR="+minSSR+" \n");
        }
        return returnval;
}
```

Segmen kode program 5. 19 Implementasi method paramMovingAverage()

```
public double[] paramAutoregressiveMovingAverage(int p,int q){
        double[] returnval=new double[p+q+1];
        double sumdata=0.0;
        for(int t=0;t<Y.length;t++){//dataYt
                sumdata=sumdata+Y[t];//dataYt
        }
        double M=sumdata/Y.length;//dataYt
        System.out.println("rata2 Y="+M);
        if(p==1){
                if(q==1){
                }
                else if(q==2){
                        double batasAtasb1=1.0,batasBawahb1=-
1.0,btsAtasb1=1.0,btsBwhb1=-1.0;
                        double b1=batasAtasb1;
                        double batasAtasm2=1,batasBawahm2=-1,btsAtasm2=1,btsBwhm2=-
1;
                        double m2=batasAtasm2;
                        double batasAtasm1=1-m2,batasBawahm1=m2-1,btsAtasm1=1-
m2,btsBwhm1=m2-1;
                        double m1=batasAtasm1;
                        double stepb1=0.1,stepm1=0.1,stepm2=0.1;
                        double minSSR = 0.0, B1=0.0, M1=0.0, M2=0.0, SSR, et_1,
et_2, Yt,Zt,et;
                        int iterke=0;
```

```
                    do{
                        batasAtasm2=1;batasBawahm2=-1;
                        m2=batasAtasm2;
                        batasAtasm1=1-m2;batasBawahm1=m2-1;
                        do{
                            do{
                                SSR=0.0;
                                et_1=0.0;
                                et_2=0.0;
                                for(int t=0;t<Y.length;t++){//dataYt
                                    if(t==0){Yt=M;}else{Yt=Y[t];}
                                    Zt=M*(1-b1)+b1*Yt-m1*et_1-
m2*et_2;//c-m1*et_1-m2*et_2;//dataYt
                                    et=Y[t]-Zt;//dataYt[t]
                                    SSR=SSR+Math.pow(et,2);
                                    et_2=et_1;
                                    et_1=et;
                                }
                                if(iterke==0){
                                    minSSR=SSR;
                                    B1=b1;
                                    M1=m1;
                                    M2=m2;
                                }
                                if(SSR<minSSR){
                                    minSSR=SSR;
                                    B1=b1;
                                    M1=m1;
                                    M2=m2;
                                    btsAtasb1=b1+stepb1;
                                    btsBwhb1=b1-stepb1;
                                    btsAtasm1=m1+stepm1;
                                    btsBwhm1=m1-stepm1;
                                    btsAtasm2=m2+stepm2;
                                    btsBwhm2=m2-stepm2;
                                }
                                m1=m1-stepm1; //m1=-0.05
                                iterke++;
                            }while(m1>batasBawahm1&&m1<batasAtasm1);
                            m2=m2-stepm2;//m2=0.95
                            batasBawahm1=batasBawahm1-stepm2;
                            batasAtasm1=batasAtasm1+stepm2;
                            m1=batasAtasm1;
                        }while(m2>batasBawahm2&&m2<batasAtasm2);
                        //START PENCARIAN DIKEDALAMAN II...VI
                        b1=b1-stepb1;
                    }while(b1>batasBawahb1&&b1<batasAtasb1);
```

```
            }
        }
        return returnval;
}
```

Segmen kode program 5. 20 Implementasi method paramAutoregressiveMovingAverage()

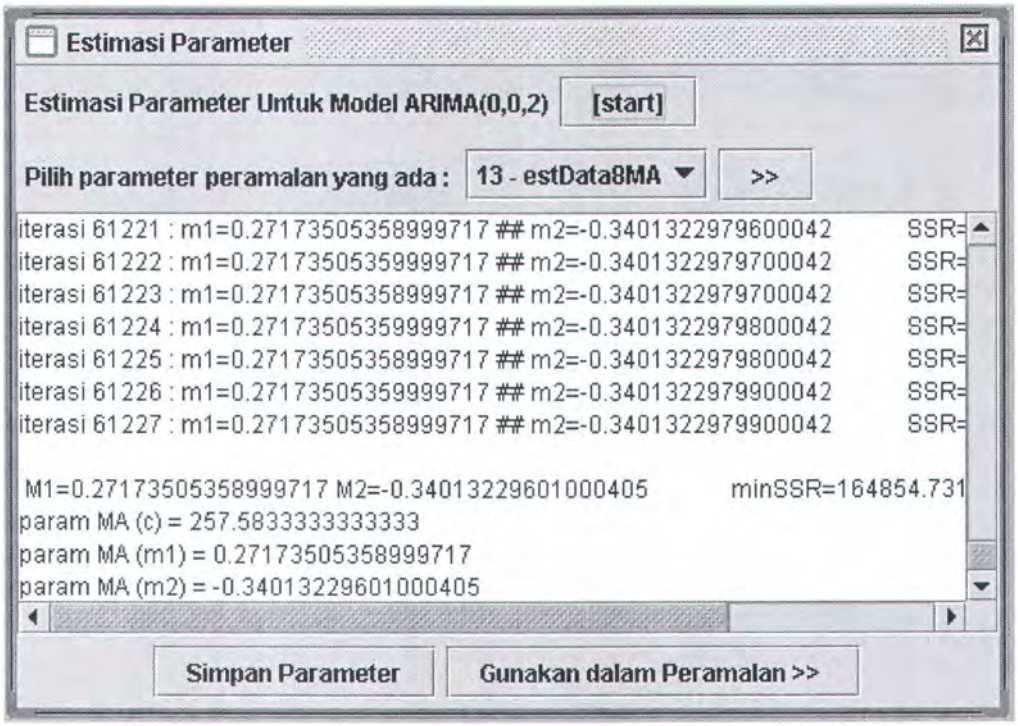
Untuk model *autoregressive* dengan order p = 2, method `paramAutoregressive()` akan memanggil method `dekomposisiCholeski()`. Method ini memberikan penyelesaian dengan aljabar matriks dalam penyelesaiannya. Method ini mengimplementasikan metode dekomposisi Choleski yang telah dijelaskan pada sub bab 2.4.3.2 di bab 2. Implementasi method ini dituliskan pada segmen kode program 5.21.

```
private double[] dekomposisiCholeski(){
        //Buat matrix X n XT
        double[] dataYtMin1=new double[dataYt.length];
        dataYtMin1=getDataYtMin(1);
        double[] dataYtMin2=new double[dataYt.length];
        dataYtMin2=getDataYtMin(2);
        double[][] X=new double[dataYt.length][3];
        double[][] XT=new double[3][dataYt.length];
        for(int t=0;t<dataYt.length;t++){
                X[t][0]=1.0;
                X[t][1]=dataYtMin1[t];
                X[t][2]=dataYtMin2[t];
                XT[0][t]=1.0;
                XT[1][t]=dataYtMin1[t];
                XT[2][t]=dataYtMin2[t];
        }
        //buat matrix XT_X
        double[][] XT_X=new double[3][3];
        for(int brs=0;brs<3;brs++){
                for(int klm=0;klm<3;klm++){
                        double isiXT_X=0.0;
                        for(int t=0;t<dataYt.length;t++){
                                isiXT_X=isiXT_X+(XT[brs][t]*X[t][klm]);
                        }
                        XT_X[brs][klm]=isiXT_X;
                }
        }
        //dekomposisi XT_X ke matrix L n LT
        double[][] LT=new double[3][3];
        LT[0][0]=Math.sqrt(XT_X[0][0]);
```

```
        LT[0][1]=XT_X[0][1]/LT[0][0];
        LT[0][2]=XT_X[0][2]/LT[0][0];
        LT[1][0]=0.0;
        LT[1][1]=Math.sqrt(XT_X[1][1]-Math.pow(LT[0][1],2.0));
        LT[1][2]=(XT_X[1][2]-LT[0][1]*LT[0][2])/LT[1][1];
        LT[2][0]=0.0;
        LT[2][1]=0.0;
        LT[2][2]=Math.sqrt(XT_X[2][2]-Math.pow(LT[0][2],2.0)-
Math.pow(LT[1][2],2.0));
        double[][] L=new double[3][3];
        L[0][0]=LT[0][0];      L[0][1]=LT[1][0];      L[0][2]=LT[2][0];
        L[1][0]=LT[0][1];      L[1][1]=LT[1][1];      L[1][2]=LT[2][1];
        L[2][0]=LT[0][2];      L[2][1]=LT[1][2];      L[2][2]=LT[2][2];
        //Buat Matrix XT_Y
        double[] XT_Y=new double[3];
        for(int brs=0;brs<3;brs++){
                double isiXT_Y=0.0;
                for(int t=0;t<dataYt.length;t++){//15
                        isiXT_Y=isiXT_Y+XT[brs][t]*dataYt[t];//Y[t];
                }
                XT_Y[brs]=isiXT_Y;
        }
        //Buat matrix Z
        double z1,z2,z3;
        z1=XT_Y[0]/L[0][0];
        z2=(XT_Y[1]-L[1][0]*z1)/L[1][1];
        z3=(XT_Y[2]-L[2][0]*z1-L[2][1]*z2)/L[2][2];
        //cari nilai utk b1,b2,b3
        double b1,b2,b3;
        b3=z3/LT[2][2];
        b2=(z2-LT[1][2]*b3)/LT[1][1];
        b1=(z1-LT[0][1]*b2-LT[0][2]*b3)/LT[0][0];
        double[] returnval={b1,b2,b3};
        return returnval;
}
```

Segmen kode program 5. 21 Implementasi method dekomposisiCholeski()

Setelah nilai-nilai parameter yang telah dianggap sesuai ditemukan, maka nilai-nilai parameter tersebut dapat disimpan dalam database. Untuk menyimpan nilai-nilai parameter tersebut pengguna dapat menekan tombol "Simpan Parameter". Method `saveParams()` dijalankan untuk menyimpan nilai-nilai parameter tersebut ke dalam database. Tampilan antarmuka untuk class `EstimasiParamForm` digambarkan oleh gambar 5.5.



Gambar 5.5 Tampilan antarmuka EstimasiParamForm

Nilai-nilai parameter hasil penaksiran tersebut akan membentuk persamaan model ARIMA. Pada tahap selanjutnya, pengguna dapat melakukan peramalan dengan menggunakan persamaan model tersebut. Pengguna diminta untuk memasukkan jumlah periode peramalan, kemudian peramalan akan dilakukan sebanyak periode tersebut. Hasil dari peramalan akan dibandingkan dengan data asli, yang diambil dari record data runtut waktu pilihan selain record data untuk observasi. Selisihnya akan digunakan untuk menghitung standard deviasi yang lebih lanjut akan digunakan untuk menghitung stok pengaman.

Untuk menuju ke tahap proses peramalan dengan persamaan model yang ditemukan, pengguna menekan tombol "Gunakan dalam Peramalan". Aksi yang dilakukan sesaat pengguna menekan tombol ini adalah terbukanya sebuah frame baru yaitu objek dari class `ForecastForm`. `ForecastForm` akan menginisialisasi beberapa komponen antara lain label `lblFormula`, *textfield* `tfPeriode`, tabel `tblForecast`, dan tabel `tblStokAman`.

Label `lblFormula` menuliskan persamaan model dengan nilai-nilai parameternya. *Textfield* `tfPeriode` akan menampung masukkan jumlah periode

peramalan yang diinginkan pengguna. Tabel `tblForecast` akan menampung hasil peramalan, data asli, dan selisih keduanya. Sedangkan tabel `tblStokAman` akan menampung nilai stok pengaman.

Proses peramalan dilakukan sesaat setelah pengguna mengisi jumlah periode peramalan dan menekan tombol ">>". Method `changeContents()` dijalankan untuk mengubah isi dari komponen `tblForecast` dan `tblStokAman` berdasarkan proses peramalan. Segmen kode program 5.22 merupakan potongan kode program yang memberikan hasil ramalan untuk model AR(1), AR(2), MA(1), dan MA(2).

```
if(p==1&&d==0&&q==0){
        hasilforecast=param[0]+param[1]*dataAsli[dataPeriode+u-1];
}else if(p==2&&d==0&&q==0){
        hasilforecast=param[0]+param[1]*dataAsli[dataPeriode+u-
1]+param[2]*dataAsli[dataPeriode+u-2];
}else if(p==0&&d==0&&q==1){
        if(u==0){
                hasilforecast=param[0]+param[1]*errorMAObsv[dataPeriode-1];
        }else{
                hasilforecast=param[0]+param[1]*error[u-1];
        }
}else if(p==0&&d==0&&q==2){
        if(u==0){
                hasilforecast=param[0]+param[1]*errorMAObsv[dataPeriode-
1]+param[2]*errorMAObsv[dataPeriode-2];
        }else if(u==1){
                hasilforecast=param[0]+param[1]*error[u-
1]+param[2]*errorMAObsv[dataPeriode-1];
        }else{
                hasilforecast=param[0]+param[1]*error[u-1]+param[2]*error[u-2];
        }
}
```

Segmen kode program 5. 22 Potongan kode program untuk peramalan dengan model AR dan MA

Hasil dari peramalan dengan persamaan model tertentu tadi selanjutnya disimpan, dan selanjutnya akan digunakan sebagai komponen pembatas pada saat melakukan perencanaan biaya penyimpanan dan lembur.

5.4 Implementasi Use Case Menjalankan Perhitungan Perencanaan Produksi

Use case Menjalankan Perhitungan Perencanaan Produksi merupakan proses perhitungan biaya penyimpanan dan biaya lembur. Perhitungan biaya penyimpanan dan biaya lembur dilakukan dengan menggunakan pemrograman linier dengan fungsi tujuan bertipe minimasi. Komponen-komponen dari pemrograman linier yang menyusun aplikasi ini, baik itu variabel-variabel keputusan, parameter-parameter, maupun modelnya, telah dibahas pada sub bab 3.4 di bab 3.

Untuk melakukan proses perhitungan ini, pengguna dapat memilih menu "Optimasi – Optimasi Komponen Biaya" pada MainFrame. Aksi yang dijalankan sesaat setelah pengguna memilih menu ini ditangani oleh method `actOpenOptimize()` dari class `FullAction`. Method `actOpenOptimize()` akan melakukan instanisasi objek dari class `OptimizeProdCostPlan`, sehingga ditampilkan sebuah frame baru.

Beberapa komponen yang diinisialisasi pada objek `OptimizeProdCostPlan` ini adalah `ComboBox cbGetProdParams`, tombol `btn_Use`, panel `formP4`, tombol `btnOptimize`, dan *textarea* `ta_LPProcess`. `ComboBox cbGetProdParams` berisi daftar data dan parameter yang dapat dipilih, dengan menekan tombol `btn_Use`, untuk digunakan sebagai parameter pada model pemrograman linier. Panel `formP4` digunakan untuk menampilkan data dan parameter yang dipilih dari `ComboBox` tadi. *Text area* `ta_LPProcess` digunakan untuk menuliskan hasil perhitungan dari iterasi algoritma simplex yang dilakukan setelah tombol `btnOptimize` ditekan.

Daftar data dan parameter yang dituliskan pada `ComboBox cbGetProdParams` diambil dari tabel `t_prodplan`. Segmen kode program 5.23 menunjukkan penulisan data dan parameter pada `ComboBox cbGetProdParams`.

```
try{
        rs=MainFrame.superkonek.select("*","t_prodplan",null);
        if(rs!=null){
                while(rs.next()){
```

```
                    cbGetProdParams.addItem(rs.getString(1)+" -
"+rs.getString(2));
                }
        }
}catch(Exception exc){exc.printStackTrace();}
```

Segmen kode program 5. 23 Penulisan data dan parameter pada ComboBox cbGetProdParams

Apabila sebuah parameter yang ada di `ComboBox` tersebut dipilih maka objek ini akan menjalankan method `openData()` yang prosesnya juga memanggil method `openData()` milik panel `formP4`. Panel `formP4` merupakan objek dari class `PanelProdPlanParams` yang berisi komponen-komponen yang akan menampilkan nilai dari data dan parameter yang telah dipilih tadi. Implementasi dari method `openData()` milik `formP4` ini dituliskan pada segmen kode program 5.24.

```
public void openData(String s){
        StringTokenizer stoken=new StringTokenizer(s,"-");
        if(stoken.hasMoreTokens()){
                idProdPlanq=stoken.nextToken();
                changeContents(idProdPlanq);
        }
}
```

Segmen kode program 5. 24 Implementasi method openData()

Method `openData()` akan memanggil method `changeContents()` yang mengubah isi dari komponen-komponen `formP4` sesuai dengan data dan parameter yang dipilih. Lebih spesifik, komponen-komponen yang diubah oleh method `changeContents()` ini adalah `dtmHolding`, `dtmProductivity`, `dtmInvAwal`, `l_PeriodeJenisProd`, `tf_Lembur`, `dtmRamalan`, `dtmSS`, dan `dtmGD`. Komponen-komponen yang berawalan huruf "dtm" merupakan objek dari `DefaultTableModel`, yang mendefinisikan struktur dari tabel. Komponen `l_PeriodeJenisProd` merupakan objek label yang menuliskan jumlah periode (T) dan jumlah jenis produk (N) yang dijadikan acuan pada jumlah baris dan kolom untuk struktur tabel `dtmGD`. Struktur tabel `dtmGD` menampilkan nilai-nilai data waktu kerja reguler, waktu kerja lembur maksimum, ramalan permintaan dan stok pengaman.

Method `changeContents()` hanya mengubah struktur tabel `dtmGD` dan mengisi kolom waktu kerja reguler dan waktu kerja lembur maksimum, namun tidak mengisi kolom ramalan permintaan dan stok pengaman. Kolom ramalan permintaan (df) dan stok pengaman (SS) masing-masing diisi dengan memilih terlebih dahulu dari daftar yang ditulis pada tabel "Hasil Ramalan Permintaan" dan tabel "Hasil Perhitungan Stok Pengaman". Method yang menangani penulisan isi kolom ramalan permintaan adalah `getDFchoosed()`, sedangkan untuk kolom stok pengaman ditangani method `getSSchoosed()`. Implementasi dari kedua method ini dituliskan berturut-turut pada segmen kode program 5.25 dan 5.26.

Setelah semua parameter telah terisi, maka pengguna dapat memulai perhitungan optimasi biaya penyimpanan dan biaya lembur dengan menekan tombol "Optimize". Aksi yang dilakukan adalah menjalankan method `showOptimationProcess()`.

```
private void getDFchoosed(){
        try{
        for(int u=0;u<jmlprod;u++){
                String id="";
                String FCchoosed=""+dtmRamalan.getValueAt(u,1);
                StringTokenizer stoken=new StringTokenizer(FCchoosed,"-");
                if(stoken.hasMoreTokens()){id=stoken.nextToken();}
                int i=0;
        rs=MainFrame.superkonek.select("*","t_forecastdata","idforecast='"+id+"'");
                if(rs!=null){
                        while(rs.next()){
                                if(i<periode){
                                        dtmGD.setValueAt(rs.getString(5),i,3+u);
                                } i++;
                        }
                }
        }
        }catch(Exception ex){ex.printStackTrace();}
}
```

Segmen kode program 5. 25 Implementasi method getDFchoosed()

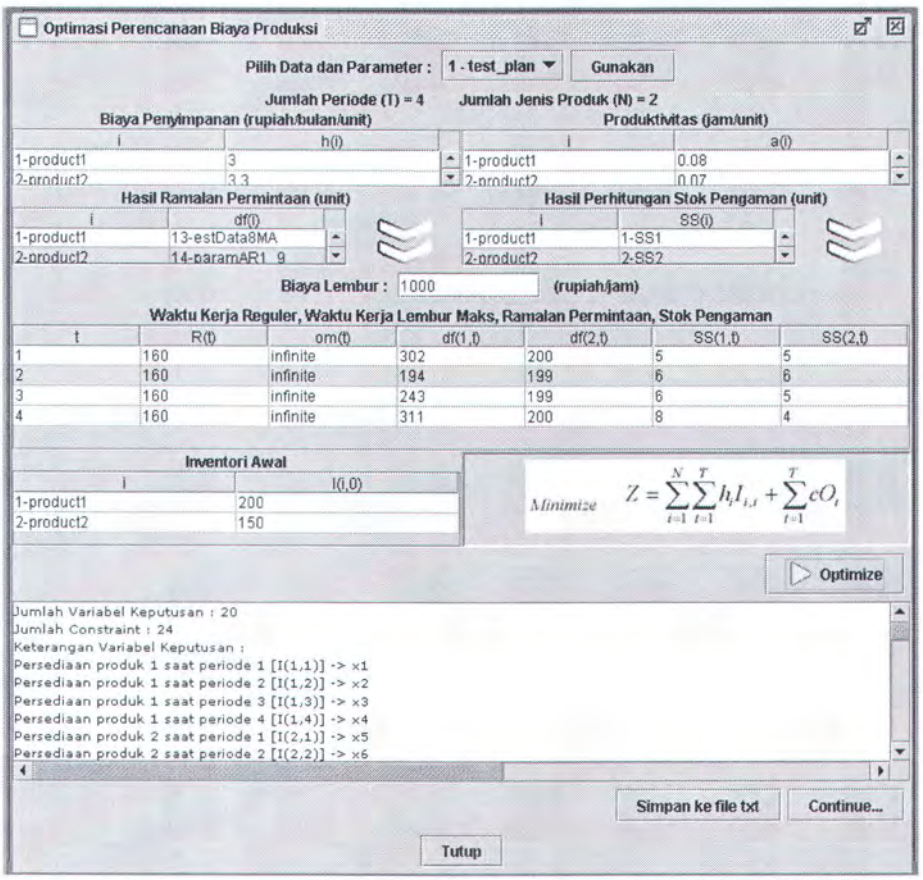
```
private void getSSchoosed(){
        try{
        for(int u=0;u<jmlprod;u++){
                String id="";
                String SSchoosed=""+dtmSS.getValueAt(u,1);
```

```
            StringTokenizer stoken=new StringTokenizer(SSchoosed,"-");
            if(stoken.hasMoreTokens()){id=stoken.nextToken();}
            int i=0;
            rs=MainFrame.superkonek.select("*","t_ssdata","idSS='"+id+"'");
            if(rs!=null){
                    while(rs.next()){
                            if(i<periode){
                            dtmGD.setValueAt(rs.getString(5),i,3+jmlprod+u);
                            } i++;
                    }
            }
        }
    }catch(Exception ex){ex.printStackTrace();}
}
```

Segmen kode program 5. 26 Implementasi method getSSchoosed()

Method `showOptimationProcess()` menjalankan method `getStartLP()` yang akan memulai proses perhitungan dengan pemrograman linier. Proses permulaan perhitungan tersebut meliputi memasukkan fungsi tujuan, memasukkan koefisien dari variabel-variabel pada fungsi pembatas, dan memasukkan sisi kanan persamaan untuk setiap pembatas. Proses pendahuluan ini melibatkan beberapa method dari class `revisedSimplex` yaitu method `specifyObjective()` untuk memasukkan fungsi tujuan dan method `addConstraint()` untuk memasukkan koefisien-koefisien pembatas dan sisi kanan persamaannya. Setelah proses pendahuluan tersebut, method `getStartLP()` menjalankan method `run()` yang akan memulai iterasi.

Method `run()` akan terus melakukan iterasi hingga nilai optimum dari fungsi tujuan tercapai. Ketika nilai optimum tercapai, iterasi akan dihentikan dan nilai optimum dari fungsi tujuan tersebut akan dituliskan beserta nilai-nilai optimum dari variabel-variabel keputusannya. Gambar 5.6 adalah tampilan antarmuka dari form perhitungan biaya penyimpanan dan biaya lembur ini.



Gambar 5.6 Tampilan antarmuka perhitungan biaya penyimpanan dan lembur

BAB VI

UJI COBA DAN ANALISIS HASIL

BAB VI

UJI COBA DAN ANALISIS HASIL

Bab ini membahas mengenai uji coba dan analisis terhadap hasil uji coba. Uji coba dilakukan dalam dua tahap yaitu verifikasi kemudian validasi. Masing-masing uji coba, baik verifikasi maupun validasi, dilakukan melalui beberapa skenario, kemudian dilakukan analisis terhadap hasil yang diperoleh. Uji coba dilakukan menggunakan lingkungan perangkat keras dan sistem operasi tertentu, serta menggunakan data tertentu.

6.1 Lingkungan Uji Coba

Semua uji coba dilakukan pada sebuah *Personal Computer* (PC) dengan prosesor AMD Sempron 1.60 GHz dan memori 480 MB RAM, serta berjalan pada sistem operasi Microsoft Windows XP Proffesional.

6.2 Data Uji Coba

Pada bab 3 telah didaftarkan dan dijelaskan parameter-parameter yang digunakan dalam perencanaan produksi. Berdasarkan tabel 3.2 terdapat sepuluh jenis parameter yang perlu diketahui nilainya dalam model perencanaan produksi. Nilai dari parameter-parameter tersebut didapatkan dari data sampel yang diambil dari PT. IGLAS (Persero). Sampel yang digunakan adalah dari jenis botol Sprite 10 oz dan Sprite 7 oz. Sebagai pelengkap, diberikan tabel 6.1 yang menunjukkan spesifikasi fisik dari jenis-jenis botol tersebut.

Tabel 6.1 Spesifikasi Fisik Sampel

Jenis Botol	Warna	Berat Rata-rata (gram)	Kapasitas (ml)
Sprite 10 oz	Hijau	380	314
Sprite 7 oz	Hijau	355	220

Setiap jenis botol dengan spesifikasi fisik yang berbeda tentunya memiliki waktu produksi (a_t) dan biaya penyimpanan persediaan (h_t) yang secara teoritis berbeda pula. Tabel 6.2 menunjukkan waktu produksi dan biaya penyimpanan persediaan secara teoritis untuk setiap jenis botol sampel.

Tabel 6.2 Waktu Produksi dan Biaya Penyimpanan Persediaan Per Jenis Botol Sampel

Jenis Botol	*a* (jam/unit)	*h* (rupiah/bulan/unit)	Perkiraan Permintaan
Sprite 10 oz	0.00000819	256	5180000
Sprite 7 oz	0.00000769	202	5180000

Parameter lainnya berkaitan dengan waktu kerja lembur dan waktu kerja reguler. Biaya lembur pegawai (c_o) diasumsikan tidak berubah setiap periodenya dan sama untuk setiap pegawainya, yaitu sebesar Rp. 50.000,00 untuk setiap jam waktu lembur. Sedangkan jumlah waktu lembur maksimum (om_t) ditetapkan sebesar 5 jam per hari atau 150 jam per bulan. Untuk waktu kerja reguler (R_t) juga diasumsikan sama untuk setiap periodenya, yaitu ditetapkan sebesar 8 jam per hari atau 176 jam per bulan, dengan asumsi sebulan 22 hari kerja.

Untuk beberapa parameter lainnya nilainya tergantung dari hasil proses peramalan, yaitu parameter hasil peramalan ($d_{t,t}$) dan stok pengaman (SS_t). Sedangkan untuk parameter jumlah persediaan awal ($I_{t,0}$), jumlah periode perencanaan (T), dan jumlah jenis produk yang digunkaan (N) ditetapkan pada saat pelaksanaan uji coba.

Selain menggunakan data sampel dari perusahaan, digunakan juga data-data dari sumber lainnya sebagai bagian dari skenario uji coba.

6.3 Skenario Uji Coba Untuk Verifikasi

Verifikasi dimaksudkan untuk menguji kesesuaian hasil didapatkan oleh aplikasi dengan hasil yang didapatkan melalui perhitungan secara manual. Uji coba akan dilakukan pada tahap-tahap metode peramalan ARIMA Box-Jenkins. Verifikasi dilakukan pada perhitungan autokorelasi dan proses *differencing*, pada estimasi parameter model ARIMA, dan pada peramalan.

6.3.1 Verifikasi Perhitungan Autokorelasi dan Proses Differencing

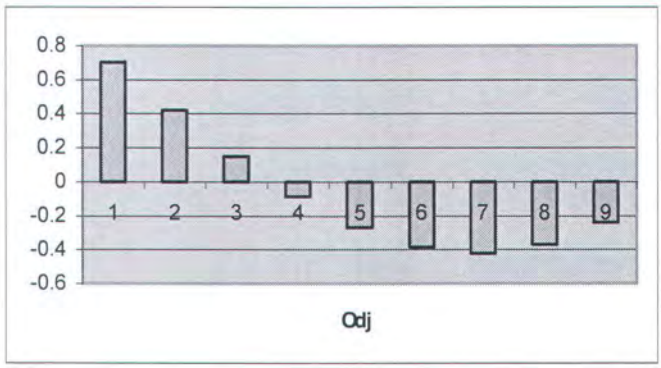
Nilai-nilai koefisien autokorelasi dan proses *differencing* berperan dalam penentuan stasioneritas data, sehingga perlu dilakukan uji kebenaran untuk hasil proses perhitungan kedua hal tersebut. Pengujian dilakukan dengan membandingkan hasil proses perhitungan aplikasi dengan hasil proses perhitungan manual. Data yang digunakan untuk uji coba adalah sampel data dalam jumlah kecil (n=10) [NGU-02], yang telah ditunjukkan oleh tabel 3.5 pada bab 3.

6.3.1.1 Pelaksanaan Uji Coba

Tabel 6.3 menunjukkan nilai koefisien autokorelasi yang dihasilkan dari perhitungan manual, dibantu perangkat lunak *spreadsheet*, Microsoft Excel. Sedangkan *correlogram* dari perhitungan autokorelasi (ACF) tersebut ditunjukkan oleh gambar 6.1.

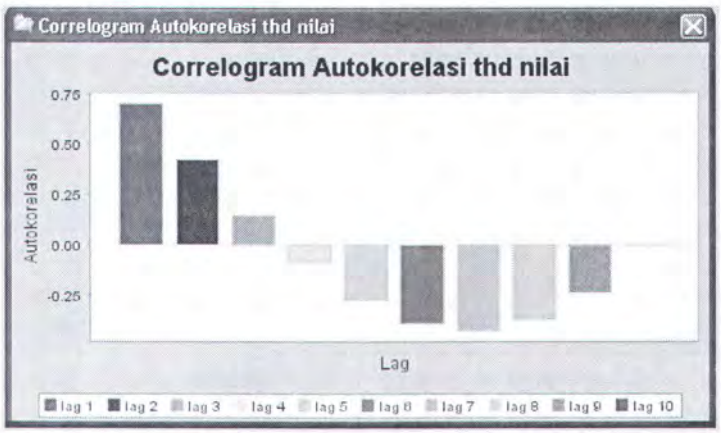
Tabel 6.3 Nilai Koefisien Autokorelasi Perhitungan Manual pada Sampel Data

k	r_k
1	0,70144
2	0,42176
3	0,14793
4	-0,08511
5	-0,26817
6	-0,38602
7	-0,42291
8	-0,36913
9	-0,23981



Gambar 6.1 Correlogram ACF Perhitungan Manual

Sedangkan pada aplikasi, nilai-nilai koefisien autokorelasi akan ditampilkan dalam bentuk *correlogram* ACF. Gambar 6.2 adalah *correlogram* ACF untuk sampel data yang ditampilkan oleh aplikasi.



Gambar 6.2 Correlogram ACF Aplikasi

Untuk verifikasi proses *differencing*, sampel data yang digunakan adalah sama dengan sampel data yang digunakan untuk verifikasi perhitungan autokorelasi. Perhitungan *differencing* secara manual untuk sampel data tersebut telah ditunjukkan oleh sub bab 3.5 pada bab 3. Sedangkan perhitungan yang dilakukan oleh aplikasi menghasilkan data-data hasil *differencing* tingkat pertama dan kedua yang ditunjukkan oleh gambar 6.3.

nilai	Diff 1	Diff 2
8		
12	4.0	
15	3.0	-1.0
19	4.0	1.0
25	6.0	2.0
30	5.0	-1.0
34	4.0	-1.0
40	6.0	2.0
45	5.0	-1.0
51	6.0	1.0

Gambar 6.3 Hasil Differencing Tingkat Satu dan Dua oleh Aplikasi

6.3.1.2 Analisis Hasil Uji Coba

Dari pelaksanaan uji coba untuk verifikasi perhitungan autokorelasi, didapatkan kenyataan bahwa hasil perhitungan autokorelasi dengan aplikasi untuk autokorelasi sama dengan hasil yang didapat oleh perhitungan manual, sehingga dapat dikatakan bahwa perhitungan autokorelasi oleh aplikasi terbukti benar.

Begitu pula untuk verifikasi perhitungan *differencing*, dari pelaksanaan uji coba didapatkan kenyataan bahwa hasil perhitungan *differencing* oleh aplikasi sama dengan hasil yang didapat oleh perhitungan manual, sehingga dapat dikatakan bahwa perhitungan *differencing* oleh aplikasi juga terbukti benar

6.3.2 Verifikasi Estimasi Parameter Model ARIMA

Estimasi parameter model ARIMA dilakukan setelah model teridentifikasi. Seperti yang telah dijelaskan pada bab 2, bahwa estimasi parameter model dilakukan untuk mendapatkan nilai minimum SSR. Untuk setiap model ARIMA, estimasi parameter yang dilakukan akan berbeda. Verifikasi dilakukan dengan membandingkan hasil estimasi parameter model oleh aplikasi dengan hasil estimasi parameter model yang telah dicapai oleh aplikasi lain yang digunakan oleh pustaka.

Dalam hal ini, sampel data yang digunakan dalam uji coba sama dengan sampel data yang digunakan oleh sub bab 3.5 pada bab 3, yaitu dua buah sampel

data [LER-02] dengan jumlah record data 100 record dan 65 record yang masing-masing dituliskan pada tabel 3.3 dan tabel 3.4.

6.3.2.1 Pelaksanaan Uji Coba

Estimasi parameter untuk sampel data pertama, yang ditunjukkan oleh tabel 3.3 adalah estimasi parameter untuk model AR(1). Identifikasi model pada sampel data pertama sehingga didapatkan model AR(1) telah dibahas sebelumnya pada sub bab 3.5. Hasil perhitungan yang didapat dari estimasi parameter terhadap model AR(1) untuk sampel data pertama tersebut adalah 0,7522 untuk koefisien ϕ_1, dan 49,3128 untuk konstanta ϕ_0.

Sedangkan estimasi parameter untuk sampel data kedua, yang ditunjukkan oleh tabel 3.4 adalah estimasi parameter untuk model MA(2). Hasil perhitungan yang diperoleh dari pembahasan estimasi parameter MA(2) di sub bab 3.5 tersebut adalah 257,380 untuk konstanta μ, 0,290 untuk koefisien ω_1, dan –0,302 untuk koefisien ω_2.

Estimasi parameter untuk sampel data pertama dengan model AR(1) yang didapatkan oleh aplikasi adalah sebagai berikut :

```
param AR (a) = 49.32586611114158
param AR (b1) = 0.7522401933164505
```

Sedangkan estimasi parameter untuk sampel data kedua dengan model MA(2) yang didapatkan oleh aplikasi adalah sebagai berikut :

```
param MA (c) = 257.38461538461536
param MA (m1) = 0.2924925751399957
param MA (m2) = -0.310318229010005
```

6.3.2.2 Analisis Hasil Uji Coba

Jika hasil estimasi parameter dengan perhitungan manual dibandingkan dengan hasil estimasi parameter oleh aplikasi, maka akan diperoleh nilai konstanta dan koefisien yang cenderung sama. Berikut adalah perbandingan hasil estimasi parameter antara perhitungan manual dan aplikasi.

Estimasi Parameter AR(1)			Estimasi Parameter MA(2)		
Manual		Aplikasi	Manual		Aplikasi
49.3128	≈	49.32586611114158	257,380	≈	257.38461538461536
0.7522	≈	0.7522401933164505	0,290	≈	0.2924925751399957
			-0,302	≈	-0.310318229010005

Karena hasil yang diperoleh estimasi parameter oleh aplikasi cenderung sama dengan estimasi parameter dengan perhitungan manual, maka estimasi parameter yang dilakukan oleh aplikasi terbukti benar.

6.3.3 Verifikasi peramalan

Peramalan dilakukan saat model terpilih telah ditemukan semua nilai konstanta dan koefisiennya. Sehingga model telah siap digunakan untuk peramalan. Verifikasi proses peramalan dilakukan dengan membandingkan hasil perhitungan aplikasi dengan hasil perhitungan manual (dibantu perangkat lunak *spreadsheet*, Microsoft Excel).

Sebagai kelanjutan dari estimasi parameter, sampel data yang digunakan untuk verifikasi peramalan ini juga diambil dari sub bab 3.5 pada bab 3. Sampel data pertama, dengan 100 record, telah diidentifikasi sebagai model AR(1) dan sampel data kedua, dengan 65 record, telah diidentifikasi sebagai model MA(2).

6.3.3.1 Pelaksanaan Uji Coba

Pada sub bab 3.5, yang melakukan perhitungan secara manual, telah didapatkan hasil peramalan untuk $t = 101$ dari sampel data pertama yaitu sebesar 200,56518. Sedangkan dari sampel data kedua, hasil peramalan untuk $t = 66$ yaitu sebesar 280,940.

Aplikasi melakukan peramalan menggunakan persamaan model hasil estimasi parameter yang telah dilakukannya. Persamaan model AR(1) untuk sampel data pertama berdasarkan hasil estimasi parameter oleh aplikasi ditunjukkan oleh persamaan 6.1. Sedangkan persamaan model MA(2) untuk sampel data kedua yang juga berdasarkan estimasi parameter oleh aplikasi ditunjukkan oleh persamaan 6.2.

$$Y_t = 49{,}32587 + 0{,}75224.Y_{t-1} \quad (6.1)$$

$$Y_t = 257{,}38462 - 0{,}29249.e_{t-1} + 0{,}31032.e_{t-2} \quad (6.2)$$

Hasil peramalan yang didapatkan aplikasi untuk sampel data pertama pada t = 101 adalah sebesar 200,586. Sedangkan untuk sampel data kedua pada t = 66 didapatkan 281,183

6.3.3.2 Analisis Hasil Uji Coba

Hasil peramalan yang didapatkan oleh aplikasi ternyata tidak berbeda secara signifikan (cenderung sama) dengan hasil peramalan yang dilakukan oleh perhitungan manual. Berikut adalah perbandingan hasil peramalan yang dilakukan secara manual dengan hasil peramalan oleh aplikasi

Hasil Peramalan Sampel Data I			Hasil Peramalan Sampel Data II		
Manual		Aplikasi	Manual		Aplikasi
200,56518	≈	200,586	280,940	≈	281,183

Karena hasil peramalan yang didapatkan oleh aplikasi cenderung sama dengan hasil peramalan yang dihitung secara manual, maka peramalan yang dilakukan oleh aplikasi terbukti benar.

6.4 Skenario Uji Coba Untuk Validasi

Validasi dilakukan untuk menguji antara keluaran aplikasi dengan hasil yang diharapkan. Uji coba ini dilakukan dalam dua buah skenario, yaitu uji coba dengan variasi model ARIMA, dan uji coba penggunaan hasil peramalan pada perencanaan produksi.

6.4.1 Skenario 1 : Uji coba dengan variasi model ARIMA

Identifikasi model dapat dilakukan oleh pengguna dengan melihat pola ACF dan PACF data. Pola ACF dan PACF data, yang digambarkan oleh correlogram, dibandingkan dengan teori yang ada, untuk ditentukan model mana

yang sesuai. Kadangkala model yang dianggap sesuai lebih dari satu, karenanya dilakukan uji coba dengan model-model ARIMA yang bervariasi.

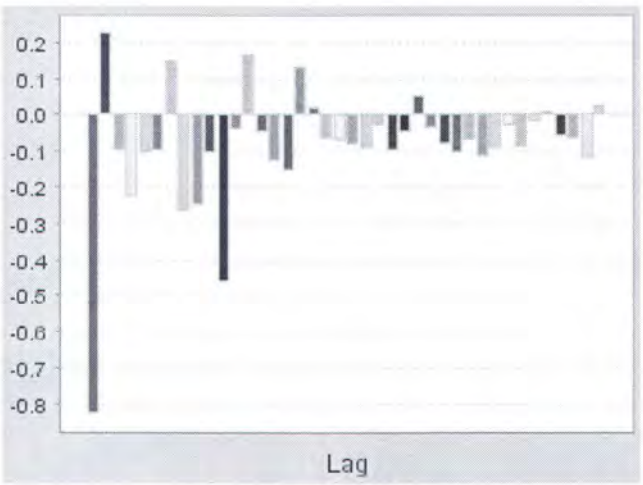
Data runtut waktu yang sama akan diuji dengan menerapkan model ARIMA yang berbeda. Uji coba dilakukan pada data pesanan botol Sprite 7 oz dengan jumlah data 50 record dan data pesanan botol Sprite 10 oz dengan jumlah data 72 record. Pada data pesanan botol Sprite 7 oz, sebanyak 40 record pertama digunakan untuk mengidentifikasi model, dan sisanya sebagai data pembanding hasil ramalan. Sedangkan pada data pesanan botol Sprite 10 oz, sebanyak 60 record pertama digunakan untuk mengidentifikasi model, dan sisanya sebagai data pembanding hasil ramalan.

Sebagai uji coba, untuk data pesanan botol Sprite 7 oz diberikan dua model yang kemudian dianggap sesuai, sebut saja model 1a dan model 1b. Sedangkan untuk data pesanan Sprite 10 oz juga diberikan dua model yang kemudian dianggap sesuai, sebut saja model 2a dan model 2b.

6.4.1.1 Identifikasi Model 1a

Identifikasi untuk model data dilakukan dengan melihat plot autokorelasi dan plot autokorelasi parsial, dengan kata lain identifikasi dilakukan dengan melihat *correlogram* ACF dan PACF, dan membandingkannya dengan *correlogram* ACF dan PACF teoritis. Gambar 6.4 dan gambar 6.5 berturut-turut menunjukkan *correlogram* ACF dan PACF data.

Gambar 6.4 Correlogram ACF Data Uji Coba (Sprite 7 oz), n = 40



Gambar 6.5 Correlogram PACF Data Uji Coba (Sprite 7 oz), n = 40

Dugaan awal berdasarkan *correlogram* ACF dan PACF pada gambar 6.4 dan 6.5 di atas, data menunjukkan model AR(2). Sehingga untuk identifikasi pertama model tersebut diidentifikasi sebagai model AR(2). Langkah berikutnya adalah melakukan estimasi parameter.

6.4.1.2 Estimasi Parameter Model 1a

Seperti yang telah dijelaskan pada bab 2, bahwa estimasi parameter untuk model AR(2) dilakukan dengan dekomposisi Choleski. Hasil dari estimasi parameter untuk data ini didapatkan sebagai berikut :

```
param AR (a) = 7458030.839259127
param AR (b1) = -0.6489527294063144
param AR (b2) = 0.2174818051806357
```

Hasil dari estimasi parameter tersebut perlu diperiksa dengan *Q-Statistic*, seperti telah dijelaskan pada bab 2.

6.4.1.3 Diagnosis Q-Statistic untuk Model 1a

Pemeriksaan (diagnosis) menggunakan *Q-Statistic* ditujukan untuk memeriksa kemungkinan adanya pola pada residual dari model yang

teridentifikasi. Apabila diagnosis *Q-Statistic* menyatakan tidak acak, maka model tersebut tidak layak untuk digunakan. Hasil dari pemeriksaan diagnosis *Q-Statistic* pada residual dari data, dengan *significant level* 0,05 dan autokorelasi yang digunakan sebanyak 30 lag pertama adalah :

```
Nilai Q Statistic untuk 30 lag pertama = 41.3409360239238
Nilai Chi-Square pada significant level 0.05 dan derajat kebebasan
30 = 43.77297183748624
Karena nilai Q Statistic LEBIH BESAR dari nilai Chi Square, maka
dapat disimpulkan residualnya TIDAK ACAK
```

Ternyata berdasarkan hasil yang didapatkan, disimpulkan residualnya tidak acak, sehingga model AR(2) tidak dapat digunakan.

6.4.1.4 Identifikasi Model 1b

Diagnosis dengan *Q-Statistic* menunjukkan model AR(2) tidak sesuai, sehingga perlu dicari model lain yang dapat dianggap lebih sesuai. Identifikasi model dilakukan dengan tetap mengacu pada *correlogram* ACF dan PACF yang didapat. Model lain yang dapat dianggap sesuai selain AR(2) adalah model ARMA(1,1).

6.4.1.5 Estimasi Parameter Model 1b

Setelah ditentukan ARMA(1,1) sebagai model yang diajukan berikutnya. Estimasi parameter untuk model ARMA(1,1) dilakukan untuk mendapatkan koefisien dan konstanta dari gabungan model AR dan MA ini. Hasil dari estimasi parameter adalah sebagai berikut :

```
param ARMA (K) = 9904804.877885146
param ARMA (b1) = -0.9016315624999983
param ARMA (m1) = -0.24212531249999947
```

6.4.1.6 Diagnosis Q-Statistic untuk Model 1b

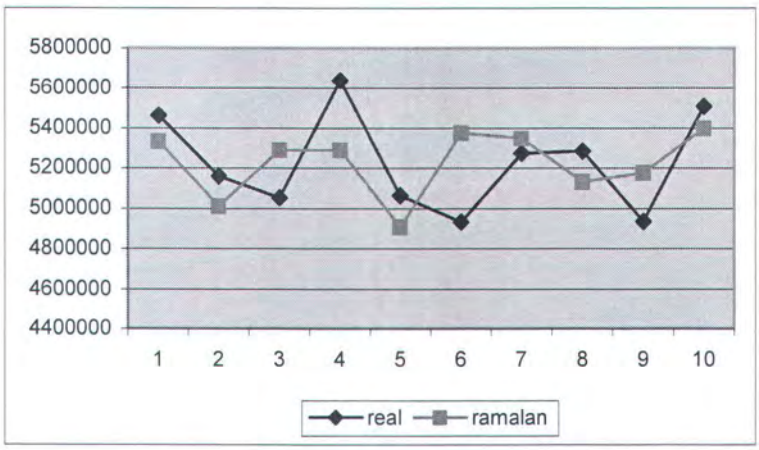
Sama dengan model yang diajukan pertama, untuk model ARMA(1,1) yang diajukan ini perlu diperiksa pula dengan *Q-Statistic*. Dari diagnosis model dengan *Q-Statistic* didapatkan hasil sebagai berikut :

```
Nilai Q Statistic untuk 30 lag pertama = 40.33497350960511
Nilai Chi-Square pada significant level 0.05 dan derajat kebebasan
30 = 43.77297183748624
Karena nilai Q Statistic LEBIH KECIL dari nilai Chi Square, maka
dapat disimpulkan residualnya ACAK
```

Ternyata berdasarkan hasil yang didapatkan, disimpulkan residualnya acak, sehingga model ARMA(1,1) dapat digunakan. Tabel 6.8 menunjukkan data *real*, hasil ramalan, dan selisih antara keduanya. Sedangkan gambar 6.6 adalah plot untuk data *real* dan hasil ramalan. Data *real* adalah 10 record terakhir dari data runtut waktu yang dipilih.

Tabel 6.4 Data Real, Hasil Ramalan 1b, dan Selisihnya

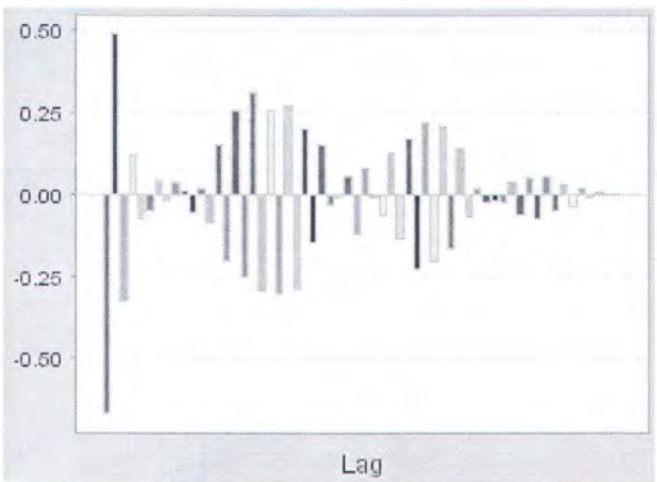
t	Real	Hasil Ramalan	Selisih
1	5462872	5332882	129990
2	5159411	5010781	148630.1
3	5053865	5288904	-235039
4	5637035	5291172	345863.3
5	5063898	4906018	157879.5
6	4932128	5377261	-445133
7	5273875	5350065	-76189.6
8	5284904	5131265	153638.7
9	4933402	5176968	-243566
10	5508167	5397720	110446.7



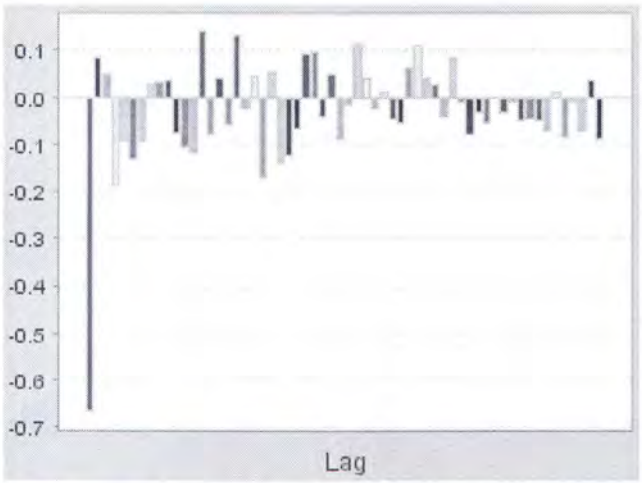
Gambar 6.6 Plot Data Real dan Hasil Ramalan Model 1b

6.4.1.7 Identifikasi Model 2a

Sama halnya dengan model 1a maupun 1b, *correlogram* ACF dan PACF digunakan untuk mengidentifikasi model. Gambar 6.7 dan gambar 6.8 berturut-turut menunjukkan *correlogram* ACF dan PACF.



Gambar 6.7 Correlogram ACF Data Uji Coba (Sprite 10 oz), n = 60



Gambar 6.8 Correlogram PACF Data Uji Coba (Sprite 10 oz), n = 60

Identifikasi pertama berdasarkan correlogram ACF dan PACF pada gambar 6.7 dan 6.8 adalah model AR(1).

6.4.1.8 Estimasi Parameter Model 2a

Estimasi parameter untuk model AR(1) yang teridentifikasi menghasilkan sebuah konstanta dan sebuah koefisien sebagai berikut :

```
param AR (a) = 8651746.236839391
param AR (b1) = -0.6684944733981653
```

Setelah dilakukan estimasi parameter dan persamaan untuk model AR(1) terbentuk. Tahap diagnosis model dengan *Q-Statistic* dilakukan.

6.4.1.9 Diagnosis Q-Statistic untuk Model 2a

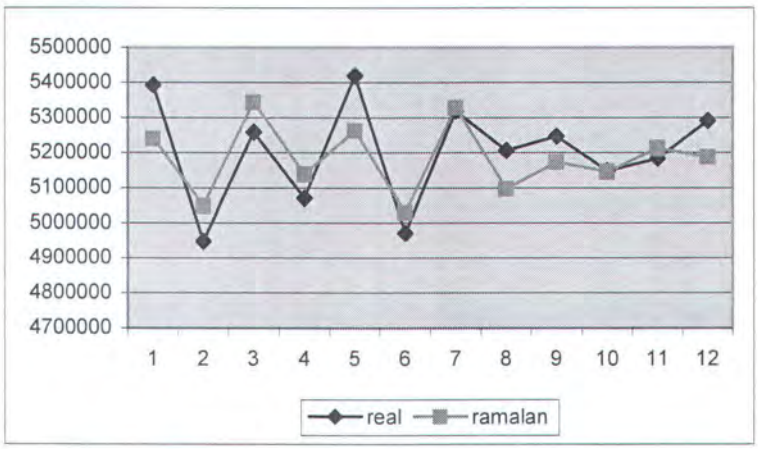
Hasil dari pemeriksaan keacakan residual untuk model AR(1) dengan *Q-Statistic* didapatkan sebagai berikut :

```
Nilai Q Statistic untuk 30 lag pertama = 27.319271162981895
Nilai Chi-Square pada significant level 0.05 dan derajat kebebasan
30 = 43.77297183748624
Karena nilai Q Statistic LEBIH KECIL dari nilai Chi Square, maka
dapat disimpulkan residualnya ACAK
```

Hasil yang didapat menyatakan bahwa residualnya acak, sehingga model AR(1) tersebut dapat digunakan. Data real dan hasil ramalan untuk 12 periode menggunakan model AR(1) tersebut, beserta selisihnya, ditunjukkan pada tabel 6.9. Sedangkan plotnya ditunjukkan pada gambar 6.9.

Tabel 6.5 Data Real, Hasil Ramalan 2a, dan Selisihnya

t	Real	Hasil Ramalan	Selisih
1	5391412	5240599	150813.2
2	4948295	5047617	-99322.1
3	5257403	5343838	-86435.4
4	5069534	5137201	-67667.4
5	5418833	5262791	156042.2
6	4971408	5029286	-57878.3
7	5317864	5328387	-10523.5
8	5205457	5096784	108673.5
9	5245235	5171927	73308
10	5146647	5145336	1311.372
11	5182674	5211241	-28567.2
12	5291093	5187157	103935.7



Gambar 6.9 Plot Data Real dan Hasil Ramalan Model 2a

6.4.1.10 Identifikasi Model 2b

Identifikasi model kedua untuk data pesanan botol Sprite 10 oz dilakukan juga dengan melihat *correlogram* ACF dan PACF pada gambar 6.7 dan gambar 6.8. Model yang diajukan adalah ARMA(1,1).

Model ARMA(1,1) merupakan gabungan model AR dan MA. Model AR(1) memiliki tingkat $p = 1$, sedangkan pada model ARMA(1,1), selain memiliki tingkat $p = 1$ pada komponen *autoregressive*, juga memiliki tingkat $q = 1$ pada komponen *moving average*. Model ARMA juga mempertimbangkan *error* masa lalu dalam persamaannya.

6.4.1.11 Estimasi Parameter Model 2b

Estimasi parameter untuk model ARMA(1,1) ini menghasilkan nilai SSR terkecil untuk konstanta dan koefisien berikut :

```
param ARMA (K) = 8814200.315274274
param ARMA (b1) = -0.7000065624999985
param ARMA (m1) = -0.07500623437500048
```

Selanjutnya, tahap diagnosis dengan *Q-Statistic* dilakukan untuk memeriksa keacakan pola residual dari model ARMA(1,1).

6.4.1.12 Diagnosis Q-Statistic untuk Model 2b

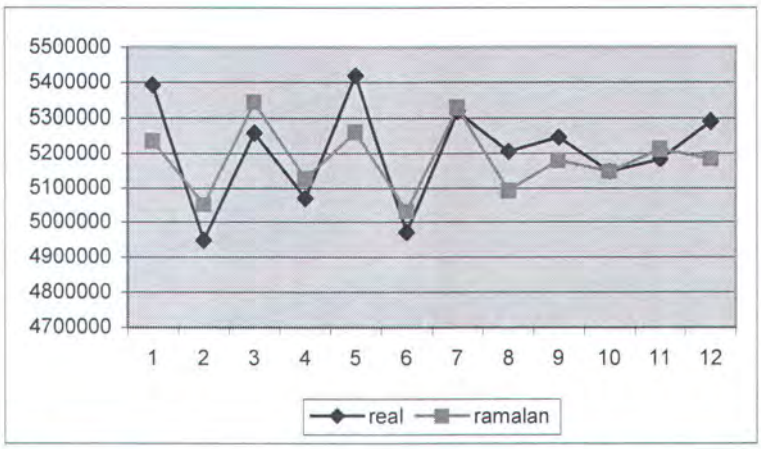
Hasil diagnosis dengan *Q-Statistic* pada model ARMA(1,1) tersebut memberikan hasil sebagai berikut :

```
Nilai Q Statistic untuk 30 lag pertama = 25.573318902665807
Nilai Chi-Square pada significant level 0.05 dan derajat kebebasan
30 = 43.77297183748624
Karena nilai Q Statistic LEBIH KECIL dari nilai Chi Square, maka
dapat disimpulkan residualnya ACAK
```

Sama seperti model AR(1), hasil yang didapat menyatakan bahwa residualnya acak, sehingga model ARMA(1,1) tersebut dapat digunakan. Data *real* dan hasil ramalan untuk 12 periode menggunakan model ARMA(1,1) tersebut, beserta selisihnya, ditunjukkan pada tabel 6.10. Sedangkan plotnya ditunjukkan pada gambar 6.10.

Tabel 6.6 Data Real, Hasil Ramalan 2b, dan Selisihnya

t	Real	Hasil Ramalan	Selisih
1	5391412	5234638	156773.9
2	4948295	5051936	-103641
3	5257403	5342588	-85184.7
4	5069534	5127594	-58060.3
5	5418833	5261138	157694.6
6	4971408	5032810	-61401.7
7	5317864	5329577	-11712.6
8	5205457	5090782	114674.9
9	5245235	5178948	66287.41
10	5146647	5147473	-826.363
11	5182674	5211452	-28777.7
12	5291093	5184136	106957



Gambar 6.10 Plot Data Real dan Hasil Ramalan Model 2b

Kedua model yang diajukan, baik AR(1) dan ARMA(1,1), ternyata diterima dalam diagnosis model menggunakan *Q-Statistic*. Bahkan plot hasil ramalan dengan model AR(1) tampak sama dengan model ARMA(1,1), pada gambar 6.9 dan gambar 6.10, walaupun terdapat perbedaan yang kecil. Dengan demikian kedua model dapat digunakan, namun peramal dapat memilih mana yang terbaik berdasarkan pertimbangan lain. Misalnya, peramal dapat juga mempertimbangkan nilai MSE (*Mean Square Error*). Nilai MSE apabila model AR(1) yang digunakan sebesar 8439474675,65 dan apabila model ARMA(1,1) yang digunakan, nilainya sebesar 8711232889,04.

6.4.2 Skenario 2 : Uji coba penggunaan hasil ramalan pada perencanaan produksi

Pada skenario ini, hasil peramalan atas suatu produk diterapkan pada perencanaan produksi. Dengan demikian dapat dilihat, bagaimana peran hasil peramalan tersebut terhadap biaya-biaya yang terjadi.

Uji coba dilakukan pada perencanaan produksi untuk 2 jenis produk dan untuk 4 periode. Kedua jenis produk yang akan diuji coba di sini adalah untuk botol Sprite 7 oz dan botol Sprite 10 oz. Waktu produksi, biaya penyimpanan, waktu kerja reguler, dan waktu kerja lembur maksimal telah diberikan pada sub bab sebelumnya.

Hasil ramalan yang digunakan adalah hasil ramalan untuk kedua jenis produk tersebut pada uji coba di skenario sebelumnya (lihat tabel 6.8 dan tabel 6.9), namun hanya diambil 4 periode pertama. Untuk botol Sprite 7 oz digunakan model ARMA(1,1), sedangkan untuk botol Sprite 10 oz digunakan model AR(1). Berikut adalah keluaran hasil proses optimasi oleh aplikasi :

```
Jumlah Variabel Keputusan : 20
Jumlah Constraint : 24
Keterangan Variabel Keputusan :
Persediaan produk 1 saat periode 1 [I(1,1)] -> x1
Persediaan produk 1 saat periode 2 [I(1,2)] -> x2
Persediaan produk 1 saat periode 3 [I(1,3)] -> x3
Persediaan produk 1 saat periode 4 [I(1,4)] -> x4
Persediaan produk 2 saat periode 1 [I(2,1)] -> x5
Persediaan produk 2 saat periode 2 [I(2,2)] -> x6
Persediaan produk 2 saat periode 3 [I(2,3)] -> x7
Persediaan produk 2 saat periode 4 [I(2,4)] -> x8
Produk 1 yang harus diproduksi saat periode 1 [X(1,1)] -> x9
Produk 1 yang harus diproduksi saat periode 2 [X(1,2)] -> x10
Produk 1 yang harus diproduksi saat periode 3 [X(1,3)] -> x11
Produk 1 yang harus diproduksi saat periode 4 [X(1,4)] -> x12
Produk 2 yang harus diproduksi saat periode 1 [X(2,1)] -> x13
Produk 2 yang harus diproduksi saat periode 2 [X(2,2)] -> x14
Produk 2 yang harus diproduksi saat periode 3 [X(2,3)] -> x15
Produk 2 yang harus diproduksi saat periode 4 [X(2,4)] -> x16
Waktu kerja lembur saat periode 1 [O(1)] -> x17
Waktu kerja lembur saat periode 2 [O(2)] -> x18
Waktu kerja lembur saat periode 3 [O(3)] -> x19
Waktu kerja lembur saat periode 4 [O(4)] -> x20

Fungsi Obyektif :
Minimize
      (202.0)x1+(202.0)x2+(202.0)x3+(202.0)x4+(256.0)x5+(256.0)x6+
(256.0)x7+(256.0)x8+(0.0)x9+(0.0)x10+(0.0)x11+(0.0)x12+(0.0)x13+(0
```

```
.0)x14+(0.0)x15+(0.0)x16+(50000.0)x17+(50000.0)x18+(50000.0)x19+(5
0000.0)x20
Koefisien Constraint
var(x1..x20) :
-1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5132882.0
1.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5010781.0
0.0  1.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5288904.0
0.0  0.0  1.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5291172.0
0.0  0.0  0.0  0.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5040599.0
0.0  0.0  0.0  0.0  1.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1.0  0.0  0.0  0.0  0.0  0.0  0.0  =  5047617.0
0.0  0.0  0.0  0.0  0.0  1.0  -1.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  1.0  0.0  0.0  0.0  0.0  0.0  =  5343838.0
0.0  0.0  0.0  0.0  0.0  0.0  1.0  -1.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  1.0  0.0  0.0  0.0  0.0  =  5137201.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  7.69E-6  0.0  0.0  0.0
8.19E-6  0.0  0.0  0.0  -1.0  0.0  0.0  0.0  <= 176.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  7.69E-6  0.0  0.0
0.0  8.19E-6  0.0  0.0  0.0  -1.0  0.0  0.0  <= 176.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  7.69E-6  0.0
0.0  0.0  8.19E-6  0.0  0.0  0.0  -1.0  0.0  <= 176.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  7.69E-6
0.0  0.0  0.0  8.19E-6  0.0  0.0  0.0  -1.0  <= 176.0
1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 310187.0
0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 310187.0
0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 310187.0
0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 310187.0
0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 191870.0
0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 191870.0
0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 191870.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  >= 191870.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  1.0  0.0  0.0  0.0  <= 150.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  1.0  0.0  0.0  <= 150.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  1.0  0.0  <= 150.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  1.0  <= 150.0


Solusi didapatkan...
NILAI OPTIMAL : 4.47105984E8
```

Nilai optimal untuk jumlah biaya penyimpanan dan biaya lembur yang dihasilkan menggunakan hasil peramalan permintaan adalah sebesar 447105984 rupiah. Sedangkan apabila perencanaan produksi dilakukan dengan jumlah permintaan berdasarkan pada intuisi yaitu untuk kedua produk sebesar kurang lebih 5180000 dan stok pengaman 300000, maka nilai optimal untuk jumlah biaya lembur dan penyimpanan sebesar 551657790 rupiah. Terdapat penghematan sekitar 104551806 rupiah.

BAB VII

KESIMPULAN

BAB VII

SIMPULAN

Sebagai penutup, bab ini akan menyimpulkan beberapa hal yang berkaitan dengan metode peramalan ARIMA Box-Jenkins maupun yang berkaitan dengan hasil yang dicapai dalam perencanaan produksi. Beberapa hal yang dapat disimpulkan berkaitan dengan metode peramalan ARIMA Box-Jenkins dan hasil yang dicapai dalam perencanaan produksi adalah sebagai berikut :

1. Identifikasi model dalam metode ARIMA Box-Jenkins terhadap suatu himpunan data runtut waktu menjadi tidak mudah apabila ciri dari model yang telah dikemukakan tidak terpenuhi secara tegas. Dimungkinkan untuk sebuah sampel data runtut waktu untuk periode tertentu memiliki model yang mendekati lebih dari satu.
2. Diagnosis terhadap residual, hanya akan memberikan keputusan suatu model ARIMA yang diajukan layak atau tidak digunakan, namun keputusan tersebut tidak dapat menyatakan model tersebut adalah yang paling sesuai, sehingga diperlukan pendekatan lain yang dapat dijadikan pertimbangan keakuratan suatu model.
3. Hasil yang dicapai oleh aplikasi dalam perencanaan produksi yang memanfaatkan hasil peramalan sebagai salah satu parameternya menunjukkan nilai jumlah biaya yang lebih kecil dibandingkan jumlah biaya yang dihasilkan tanpa memanfaatkan hasil peramalan. Sehingga penggunaan hasil peramalan lebih baik daripada pertimbangan subyektif atau intuisi saja.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

1) [BOW-05] Bruce L. Bowerman, Richard T. O'Connel, Anne B. Koehler, "Forecasting, Time Series and Regression", Thomson Brooks/Cole, 2005.
2) [GAS-02] Vincent Gaspersz, "Production Planning And Inventory Control". PT. Gramedia Pustaka Utama Jakarta, 2002.
3) [HAM-03] Hamdy A. Taha, "Operation Research: An Introduction", Pearson Education, Inc. 2003.
4) [HAN-01] John E. Hanke, Arthur G. Reitsch, Dean W. Wichern. "Business Forecasting", Prentice Hall, Inc. 2001.
5) [LER-02] Lerbin R. Aritonang R., "Peramalan Bisnis", Ghalia Indonesia, 2002.
6) [MAK-98] Spyros Makridakis, Steven C. Wheelwright, Victor E. McGee. "Forecasting : Methods and Application" John Wiley & Sons, Inc. 1998.
7) [NGU-02] Ngurah Agus Sanjaya. "Perancangan dan Pembuatan Perangkat Lunak Peramalan Data Time Series Menggunakan Metode Box Jenkins" Tugas Akhir, Jurusan Teknik Informatika FTIF ITS Surabaya. 2002.
8) [PIS-01] Pisal Yenradee, Amnaj Charoenthavornying, Anulark Pinnoi. "Demand Forecasting and Production Planning for Highly Seasonal Demand Situations : Case Study of a Pressure Container Factory" in ScienceAsia 27, 271-278. 2001.
9) [QUA-02] Terry Quatrani. "Analysis and Design with UML" Rational Software Corporation. 2002
10) [SAI-00] Ahmad Saikhu. "Perbandingan Metode Neural Network dan Metode ARIMA untuk Peramalan Data Time Series" Tesis, Program Pasca Sarjana Teknik Informatika, ITS Surabaya. 2000.
11) [TJU-03] Tjutju Tarliah Dimyati, Ahmad Dimyati, "Operation Research Model-Model Pengambilan Keputusan", Sinar Baru Algensindo, 2003.
12) [WAL-89] Ronald E. Walpole and Raymond H. Myers, "Probability and Statictics for Engineers and Scientists", Macmillan Publishing Co, Inc. 1989.