

POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea Triennale in Ingegneria Gestionale classe L-8



Previsione della domanda, MPS, ATP e simulazione ordini extra

Relatore:

Prof. Fulvio Corno

Studente:

Rebecca Pelacà

Matricola s218118

Indice

1	Proposta di progetto	2
1.1	Descrizione del problema proposto	2
1.2	Descrizione della rilevanza gestionale del problema	2
1.3	Descrizione dei data-set per la valutazione	2
1.4	Descrizione preliminare degli algoritmi coinvolti	2
1.5	Descrizione preliminare delle funzionalità previste per l'applicazione software	2
2	Descrizione dettagliata del problema affrontato	3
2.1	Forecasting	3
2.2	MPS e ATP	5
2.3	Simulazione	6
3	Data-set	6
4	Strutture dati e algoritmi	7
4.1	Model	7
4.2	Simulazione	12
5	Diagramma delle classi principali	13
6	Esempio di svolgimento	14
6.1	Esempio : forecasting	14
6.2	Esempio : MPS e ATP	14
6.3	Esempio : simulazione	15
7	Conclusioni	16

1 Proposta di progetto

1.1 Descrizione del problema proposto

L'applicazione si propone di fornire un modo semplice e veloce per svolgere alcuni dei calcoli più comuni in ambito di produzione aziendale, ossia quello della previsione della domanda futura con conseguente piano principale di produzione. Inoltre, si vuole implementare un modo per simulare futuri ordini extra e avere statistiche sulle quantità di ordini accettati o rifiutati. I risultati della simulazione, interpretati correttamente in base alle esigenze dell'azienda, potrebbero aiutare nella decisione di mantenere il piano di produzione o apportare modifiche opportune.

1.2 Descrizione della rilevanza gestionale del problema

Il problema affronta temi caratteristici della gestione della produzione aziendale. Mediante il data-set dello storico della domanda degli ultimi anni, le formule di *forecasting* e gli eventuali parametri inseriti dall'utente in caso di trend e/o stagionalità, sarà possibile avere previsioni sulla domanda futura, indispensabile per definire il *Master Production Schedule*. L'MPS, difatti, definisce le quantità di prodotti finiti che devono essere prodotte in specifici intervalli di tempo. E' previsto anche il calcolo dell'*Available To Promise* (ATP), che definisce le quantità di prodotti finiti che possono essere consegnate in un determinato periodo, utile per capire se un ordine extra può essere accettato o meno. Infatti, un ordine può essere accettato se non rende negativo l'ATP.

1.3 Descrizione dei data-set per la valutazione

Il data-set aziendale è stato fornito dal docente di Programmazione e Gestione della Produzione e consiste in un archivio dello storico triennale della domanda di quattro prodotti diversi.

1.4 Descrizione preliminare degli algoritmi coinvolti

L'applicazione implementerà una serie di formule per il calcolo dei componenti da stampare poi in forma tabellare. Verrà utilizzato un algoritmo di simulazione: verrà associata opportunamente ad ogni intervallo di tempo una certa probabilità di ricevere ordini extra e sarà controllata la loro fattibilità.

1.5 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'idea è quella di sviluppare un'applicazione in linguaggio Java che permetta all'utente di:

- Selezionare un prodotto e un algoritmo opportuno in base a stagionalità e/o trend e quindi inserire l'orizzonte temporale τ (per un massimo di 9 mesi) e gli eventuali parametri m , α , β , γ , N . Inseriti tutti i dati, sarà possibile procedere con il calcolo della previsione della domanda;

- Inserire *booked orders* (per un massimo di 9 mesi), *MPS lot-size*, e l'eventuale disponibilità iniziale del magazzino. Dopodichè sarà possibile procedere con il calcolo di MPS e ATP;
- Inserire un valore di probabilità di ricevere ordini extra, da attribuire ad ogni *time bucket* (nella fattispecie si tratta di mesi) e un possibile range di quantità di prodotto richiesta. Sarà possibile in questo modo lanciare l'algoritmo di simulazione in modo da avere statistiche sulle quantità di ordini che si potrebbero accettare/rifiutare e prendere decisioni su eventuali modifiche nel piano di produzione.

2 Descrizione dettagliata del problema affrontato

L'applicazione affronta alcune tra le più comuni problematiche aziendali. Si tratta di voler fare una previsione della domanda futura, calcolare il piano principale di produzione e la disponibilità effettiva del prodotto, e simulare ordini extra per verificare la flessibilità del piano.

In seguito verranno analizzate nel dettaglio le tre fasi in cui si struttura l'applicazione.

2.1 Forecasting

La fase di *forecasting*, ossia di previsione della domanda, necessita di alcuni dati di input:

- Lo storico della domanda di un prodotto, di cui l'azienda tiene traccia in un data-set opportuno;
- Un valore τ che rappresenta l'orizzonte temporale di cui voglio avere la previsione;
- Un metodo di previsione adeguato per l'andamento della domanda. E' necessario, infatti, che prima di procedere l'azienda si occupi di fare con un'analisi dello storico in modo da evidenziare eventuali fattori di trend e/o stagionalità. I metodi previsti per questa applicazione sono i seguenti.

Moving average Metodo adatto in assenza di trend. E' attribuito lo stesso peso per le ultime m osservazioni.

$$F(t) = \frac{\sum_{i=t-m+1}^t A(i)}{m}$$

$$f(t + \tau) = F(t)$$

Exponential Smoothing Metodo adatto in assenza di trend. Il peso attribuito diminuisce esponenzialmente per le osservazioni passate.

$$F(t) = \alpha A(t) + (1 - \alpha)F(t - 1)$$

$$f(t + \tau) = F(t)$$

Exponential Smoothing with trend Metodo adatto in presenza di trend lineari. Il peso attribuito diminuisce esponenzialmente per le osservazioni passate.

$$F(t) = \alpha A(t) + (1 - \alpha)[F(t - 1) + T(t - 1)]$$

$$T(t) = \beta[F(t) - F(t - 1)] + (1 - \beta)T(t - 1)$$

$$f(t + \tau) = F(t) + \tau T(t)$$

Winter's method with multiplicative seasonal factors Metodo adatto in presenza di serie stagionali che si ripetono ogni N periodi, $N \geq 3$. I fattori stagionali sono rappresentati da un set di moltiplicatori c_t che rappresentano la quantità media di cui la domanda differisce dalla media complessiva.

$$F(t) = \alpha \left[\frac{A(t)}{c(t - N)} \right] + (1 - \alpha)[F(t - 1) + T(t - 1)]$$

$$T(t) = \beta[F(t) - F(t - 1)] + (1 - \beta)T(t - 1)$$

$$c(t) = \gamma \left[\frac{A(t)}{F(t)} \right] + (1 - \gamma)c(t - N)$$

$$f(t + \tau) = [F(t) + \tau T(t)]c(t + \tau - N)$$

Winter's method with additive seasonal factors Metodo adatto in presenza di serie stagionali che si ripetono ogni N periodi, con $N \geq 3$. I fattori stagionali sono rappresentati da un set di moltiplicatori c_t che rappresentano la quantità media di cui la domanda differisce dalla media complessiva.

$$F(t) = \alpha[A(t) - c(t - N)] + (1 - \alpha)[F(t - 1) + T(t - 1)]$$

$$T(t) = \beta[F(t) - F(t - 1)] + (1 - \beta)T(t - 1)$$

$$c(t) = \gamma[A(t) - F(t - 1) - T(t - 1)] + (1 - \gamma)c(t - N)$$

$$f(t + \tau) = F(t) + \tau T(t) + c(t + \tau - N)$$

In output sarà disponibile la previsione della domanda per l'intervallo di tempo definito (τ).

2.2 MPS e ATP

La seconda fase consiste nel calcolo di MPS e ATP.

Il Master Production Schedule in un sistema Push definisce le quantità di end-items che devono essere prodotte in specifici intervalli di tempo. Gli end-items, a seconda della diversa strategia aziendale utilizzata, possono essere prodotti finiti, sottoassemblati o materie prime. I dati in input necessari per procedere sono i seguenti:

- $F(t)$ - la previsione della domanda per un determinato intervallo di tempo, suddiviso in time buckets;
- $O(t)$ - gli ordini acquisiti per i corrispondenti time buckets;
- $MPSLotSize$ - le quantità di end-item da mandare in produzione quando l'algoritmo lo richiede;
- I_0 - la disponibilità iniziale del magazzino.

Scopo dell'MPS è mantenere non negativo il *Projected on-hand inventory* individuando quando e quanto produrre.

In output, per ogni time bucket, l'algoritmo calcolerà:

- I_t - il Projected on-hand inventory: fornisce una stima delle scorte disponibili in ogni periodo dopo che è stata soddisfatta la domanda di quel periodo.

$$I_t = I_{t-1} + MPS_t - \max[F_t, O_t]$$

- MPS_t - la previsione della quantità da produrre nel periodo t. Occorre inserire una quantità MPS ogni volta che il Projected on-hand inventory diventa negativo.

Inoltre, è possibile calcolare, in corrispondenza di ogni quantità MPS, l'*Available to Promise*. Questo definisce le quantità di prodotti finiti che possono essere consegnate in un determinato periodo; si tratta di un modo per determinare se soddisfare o meno gli ordini extra. Un ordine, infatti, può essere accettato se non rende negativo l'ATP.

Primo periodo:

$$ATP_1 = I_0 + MPS_1 - \sum_{t'=1}^n O_{t'}$$

Altri periodi:

$$ATP_t = MPS_t - \sum_{t'=t}^n O_{t'}$$

con $n = \text{next } MPS_t$

Se il risultato del calcolo in un certo time bucket esce negativo, l'ATP è impostato a 0 e la quantità negativa è sottratta in valore assoluto all'ATP precedente.

2.3 Simulazione

La terza e ultima fase consiste nella simulazione di ordini extra futuri, attraverso un algoritmo appropriato, al fine di avere una stima delle quantità di prodotto richiesto accettabili e non. La simulazione consente di verificare la flessibilità del piano agli ordini aggiuntivi, e può essere uno strumento per decidere se lasciarlo inalterato o apporre modifiche opportune.

In input l'algoritmo necessita di:

- p - un valore p di probabilità di ricevere ordini aggiuntivi, da attribuire ad ogni time bucket;
- $[max, min]$ - un range di valori entro il quale si stima di collocare i possibili ordini in ogni time bucket.

In output l'applicazione presenterà:

- il mese in cui si verifica l'ordine simulato;
- la quantità di prodotto richiesta dall'ordine simulato e se l'ordine simulato è accettato oppure no;
- il numero totale di ordini simulati accettati e di quelli rifiutati;
- il valore percentuale di ordini simulati accettati e rifiutati.

3 Data-set

Il data-set aziendale utilizzato consiste nello storico dal 01/01/2014 al 20/10/2017 della domanda di quattro prodotti diversi. I dati, forniti tramite un file Excel, sono stati rielaborati e inseriti in un database SQL. All'interno del progetto di tesi sono inseriti: il package "data-set" con il codice utilizzato per creare le tabelle e riempire il database, il file .sql con i dati già inseriti, e i file originali.

Di seguito i frammenti di codice per la creazione delle tabelle e l'inserimento dei dati.

```
String createTableProducts = "CREATE TABLE products ("
    + "productId varchar(20), "
    + "PRIMARY KEY (productId)"
    + ");"
;

String createTableStorico = "CREATE TABLE storico ("
    + "contatore int, "
    + "productId varchar(20), "
    + "date date, "
    + "x char(10), "
    + "quantity int, "
    + "PRIMARY KEY (contatore, productId), "
    + "FOREIGN KEY (productId) REFERENCES products(productId)"
    + ");"
;
```

```

String p1 = "C:\\Users\\Rebecca\\Desktop\\POLITO\\TDP\\Workspace\\DATA-SET\\prod1.csv";
String p2 = "C:\\Users\\Rebecca\\Desktop\\POLITO\\TDP\\Workspace\\DATA-SET\\prod2.csv";
String p3 = "C:\\Users\\Rebecca\\Desktop\\POLITO\\TDP\\Workspace\\DATA-SET\\prod3.csv";
String p4 = "C:\\Users\\Rebecca\\Desktop\\POLITO\\TDP\\Workspace\\DATA-SET\\prod4.csv";

String prod1 = "ZPS0001";
String prod2 = "ZPS0002";
String prod3 = "ZPS0003";
String prod4 = "ZPS0004";

Map<String, String> file = new HashMap<String, String>();

file.put(p1, prod1);
file.put(p2, prod2);
file.put(p3, prod3);
file.put(p4, prod4);

for(String key : file.keySet()) {

    List<String> lines = new ArrayList<String>();

    lines = Data_set.LoadLines(key);

    String insertProd = "INSERT INTO products(productId) "
        + "VALUES(?);";

    String insertX = "INSERT INTO storico(contatore, productId, date, x, quantity) "
        + "VALUES(?, ?, ?, ?, ?);";
}

```

4 Strutture dati e algoritmi

Le principali strutture dati e gli algoritmi si trovano nella classe Model del package tesi.model del progetto. Fa eccezione l'algoritmo di simulazione, per il quale è stata aggiunta una classe apposita, all'interno dello stesso package, chiamata appunto Simulazione.

Un altro package è stato dedicato alla connessione al database (classe DBConnect) e all'estrazione tramite query SQL dei dati richiesti dal model (classe DAO).

Infine, il main, i metodi di interazione con l'interfaccia grafica e il file .fxml, fanno parte del package tesi.

Nelle sezioni successive, una descrizione ad alto livello delle principali strutture dati e degli algoritmi implementati.

4.1 Model

La classe Model presenta i seguenti attributi:

- Un'istanza della classe DAO, per ricavare i dati del DB utili per l'implementazione.
- Una lista di interi per la previsione della domanda, attributo di Model poiché necessaria per diversi metodi della classe.
- Un array di interi per la definizione dell'ATP nei vari time buckets, attributo di Model per un motivo analogo.

Le altre strutture dati sono dichiarate all'interno dei metodi della classe. Di seguito quindi un elenco dei metodi e, per i più importanti, sono allegati i frammenti chiave del codice corrispondente:

- *getProdotti()* - Metodo che restituisce una lista dei prodotti inserita in un menù a tendina dell'interfaccia, da cui l'utente può scegliere il prodotto su cui iniziare le analisi.
- *getSeries()* - Metodo che restituisce un oggetto di tipo Series, necessario per la rappresentazione su assi cartesiani dell'andamento della domanda del prodotto selezionato. L'asse x è la linea del tempo, suddiviso in intervalli mensili. Sull'asse y è rappresentata la domanda per il mese corrispondente.
- *getMetodi()* - Metodo che restituisce una lista dei cinque metodi di previsione disponibili, inseriti in un menù a tendina dell'interfaccia.
- *getMovingAverage()*, *getExponentialSmoothing()*, *getExponentialSmoothingWithTrend()*, *getWinterMult()*, *getWinterAdd()* - Metodi che restituiscono un ObservableList con la previsione della domanda per l'intervallo di tempo τ richiesto.

E' necessario salvare i dati in questo tipo di lista per poterli poi visualizzare in una tabella nell'interfaccia grafica. Ai fini del progetto, è stato dato un massimo valore di τ equivalente a 9 mesi, stimando di non poter avere previsioni affidabili per numeri più alti. I metodi sono stati implementati seguendo le formule della sottosezione 2.1.

```
public ObservableList<Forecast> getMovingAverage(Prodotto prodotto, int tau, int m) throws ArithmeticException{
    this.forecast = new ArrayList<Integer>();
    ObservableList<Forecast> result = FXCollections.observableArrayList();
    List<Integer> demand = dao.getStoricoDB(prodotto);
    List<Double> smoothed_estimate = new ArrayList<Double>();
    int tot;
    double media;
    for(int i=m-1; i<demand.size(); i++) {
        tot = 0;
        media = 0;
        for(int j=0; j<m; j++) {
            tot += demand.get(i-j);
        }
        if(m==0)
            throw new ArithmeticException();
        media = (double)tot/m;
        smoothed_estimate.add(media);
    }
    for(int i=tau-1; i>=0; i--) {
        int last_index = smoothed_estimate.size()-1;
        forecast.add((int)Math.round(smoothed_estimate.get(last_index-i)));
    }
}
```

getMovingAverage()

```

public ObservableList<Forecast> getExponentialSmoothing(Prodotto prodotto, int tau, double alfa) {
    ObservableList<Forecast> result = FXCollections.observableArrayList();

    List<Integer> demand = dao.getStoricoDB(prodotto);
    List<Double> smoothed_estimate = new ArrayList<Double>();
    this.forecast = new ArrayList<Integer>();

    double f;

    smoothed_estimate.add((double)demand.get(0));

    for(int i=1; i<demand.size(); i++) {
        f = alfa*demand.get(i)+(1-alfa)*smoothed_estimate.get(i-1);
        smoothed_estimate.add(f);
    }

    for(int i=tau-1; i>=0; i--) {
        int last_index = smoothed_estimate.size()-1;
        forecast.add((int)Math.round(smoothed_estimate.get(last_index-i)));
    }
}

```

getExponentialSmoothing()

```

public ObservableList<Forecast> getExponentialSmoothingWithTrend(Prodotto prodotto, int tau, double alfa, double beta) {
    ObservableList<Forecast> result = FXCollections.observableArrayList();

    List<Integer> demand = dao.getStoricoDB(prodotto);
    List<Double> smoothed_estimate = new ArrayList<Double>();
    List<Double> smoothed_trend = new ArrayList<Double>();
    this.forecast = new ArrayList<Integer>();

    double f;
    double t;

    smoothed_estimate.add((double)demand.get(0));
    smoothed_trend.add(0.0);

    for(int i=1; i<demand.size(); i++) {
        f = alfa*demand.get(i) + (1-alfa)*(smoothed_estimate.get(i-1)+smoothed_trend.get(i-1));
        smoothed_estimate.add(f);
        t = beta*(smoothed_estimate.get(i)-smoothed_estimate.get(i-1))+(1-beta)*smoothed_trend.get(i-1);
        smoothed_trend.add(t);
    }

    for(int i=tau-1; i>=0; i--) {
        int last_index = smoothed_estimate.size()-1;
        forecast.add((int)Math.round(smoothed_estimate.get(last_index-i)+tau*smoothed_trend.get(last_index-i)));
    }
}

```

getExponentialSmoothingWithTrend()

```

public ObservableList<Forecast> getWinterMult(Prodotto prodotto, int tau, double alfa, double beta, double gamma, int N) throws ArithmeticException {
    ObservableList<Forecast> result = FXCollections.observableArrayList();

    List<Integer> demand = dao.getStoricoDB(prodotto);
    List<Double> smoothed_estimate = new ArrayList<Double>();
    List<Double> smoothed_trend = new ArrayList<Double>();
    List<Double> smoothed_seasonality = new ArrayList<Double>();
    this.forecast = new ArrayList<Integer>();

    int sum = 0;
    double average;
    for(int i=0; i<N; i++)
        sum += demand.get(i);
    average = (double)sum/N;

    if(average==0)
        throw new ArithmeticException();

    for(int i=0; i<N; i++) {
        smoothed_seasonality.add(demand.get(i)/average);
        smoothed_trend.add(0.0);
        if(i<N-1)
            smoothed_estimate.add(0.0);
    }

    smoothed_estimate.add(average);
}

```

getWinterMult() pt1

```

double f;
double t;
double c;

for(int i=N; i<demand.size(); i++) {
    if(smoothed_seasonality.get(i-N)==0 || smoothed_estimate.get(i)==0)
        throw new ArithmeticException();
    f = alfa*(demand.get(i)/smoothed_seasonality.get(i-N))+(1-alfa)*(smoothed_estimate.get(i-1)+smoothed_trend.get(i-1));
    smoothed_estimate.add(f);
    t = beta*(smoothed_estimate.get(i)-smoothed_estimate.get(i-1))+(1-beta)*smoothed_trend.get(i-1);
    smoothed_trend.add(t);
    c = gamma*(demand.get(i)/smoothed_estimate.get(i))+(1-gamma)*smoothed_seasonality.get(i-N);
    smoothed_seasonality.add(c);
}

for(int i=tau-1; i>=0; i--) {
    int last_index = smoothed_estimate.size()-1;
    int res = (int)Math.round((smoothed_estimate.get(last_index-i)+tau*smoothed_trend.get(last_index-i))*smoothed_seasonality.get(last_index-i-N+1));
    forecast.add(res);
}

```

getWinterMult() pt2

```

public ObservableList<Forecast> getWinterAdd(Prodotto prodotto, int tau, double alfa, double beta, double gamma, int N) throws ArithmeticException {
    ObservableList<Forecast> result = FXCollections.observableArrayList();

    List<Integer> demand = dao.getStoricoDB(prodotto);
    List<Double> smoothed_estimate = new ArrayList<Double>();
    List<Double> smoothed_trend = new ArrayList<Double>();
    List<Double> smoothed_seasonality = new ArrayList<Double>();
    this.forecast = new ArrayList<Integer>();

    int sum = 0;
    double average;

    for(int i=0; i<N; i++)
        sum += demand.get(i);
    average = (double)sum/N;

    if(average==0)
        throw new ArithmeticException();

    for(int i=0; i<N; i++) {
        smoothed_seasonality.add(demand.get(i)/average);
        smoothed_trend.add(0.0);
        if(i<N-1)
            smoothed_estimate.add(0.0);
    }

    smoothed_estimate.add(average);
}

```

getWinterAdd() pt1

```

double f;
double t;
double c;
for(int i=N; i<demand.size(); i++) {
    f = alfa*(demand.get(i)-smoothed_seasonality.get(i-N))+(1-alfa)*(smoothed_estimate.get(i-1)+smoothed_trend.get(i-1));
    smoothed_estimate.add(f);
    t = beta*(smoothed_estimate.get(i)-smoothed_estimate.get(i-1))+(1-beta)*smoothed_trend.get(i-1);
    smoothed_trend.add(t);
    c = gamma*(demand.get(i)-smoothed_estimate.get(i-1)-smoothed_trend.get(i-1))+(1-gamma)*smoothed_seasonality.get(i-N);
    smoothed_seasonality.add(c);
}

for(int i=tau-1; i>=0; i--) {
    int last_index = smoothed_estimate.size()-1;
    int res = (int)Math.round(smoothed_estimate.get(last_index-i)+tau*smoothed_trend.get(last_index-i)+smoothed_seasonality.get(last_index-i-N+1));
    forecast.add(res);
}

```

getWinterAdd() pt2

- *getMPSeATP()* - Metodo che restituisce un ObservableList con i risultati del calcolo di MPS e ATP. Anch'essi vengono poi visualizzati in forma tabellare nell'interfaccia grafica. Per questo metodo sono state implementate le formule già presentate nella sottosezione 2.2.

```

for(int i=0; i<forecast.size(); i++) {

    It = 0;

    if(i==0)
        It = magIn + MPSQuantity[i];
    else
        It = disponibilita_magazzino[i-1] + MPSQuantity[i];

    if(forecast.get(i)>=ordini_acquisiti[i])
        It -= forecast.get(i);
    else
        It -= ordini_acquisiti[i];

    if(It>=0)
        disponibilita_magazzino[i] = It;
    else {
        MPSQuantity[i] = lotSize;

        if(i==0)
            It = magIn + MPSQuantity[i];
        else
            It = disponibilita_magazzino[i-1] + MPSQuantity[i];

        if(forecast.get(i)>=ordini_acquisiti[i])
            It -= forecast.get(i);
        else
            It -= ordini_acquisiti[i];

        disponibilita_magazzino[i] = It;
    }
}

```

frammento del calcolo dell'MPS

```

for(int i=0; i<forecast.size(); i++) {

    atp = 0;
    calcolaATP = false;

    if(i==0) {
        atp = magIn + MPSQuantity[i];
        calcolaATP = true;
    }
    else if(MPSQuantity[i]>0) {
        atp = MPSQuantity[i];
        calcolaATP = true;
    }
    if(calcolaATP == true) {
        for(int j=i; j<MPSQuantity.length; j++) {
            if(j==i || MPSQuantity[j]==0)
                atp -= ordini_acquisiti[j];
            else if(MPSQuantity[j]>0)
                break;
        }
        if(atp<0 && i!=0) {
            for(int j=i; j>=0; j--) {
                if(ATP[j]>-atp) {
                    ATP[j] += atp;
                    ATP[i] = 0;
                    break;
                }
                else if (ATP[j]<-atp && ATP[j]>0) {
                    atp += ATP[j];
                    ATP[j] = 0;
                    ATP[i] = 0;
                }
            }
        }
        else
            ATP[i] = atp;
    }
}

```

frammento del calcolo dell'ATP

- *simulaModel()* - Metodo che restituisce una stringa con i risultati della simulazione. Il metodo sfrutta una certa probabilità p , inserita dall'utente, che

in ogni time bucket (cioè in ogni mese) si riceva un certo numero, estratto randomicamente tra un massimo e un minimo inseriti anch'essi dall'utente, di ordini aggiuntivi.

Creata un'istanza della classe Simulazione, per ogni time bucket in cui la probabilità prevede una quantità di prodotto extra, il metodo aggiunge l'ordine alla coda degli eventi, tramite il metodo *addOrdine()* della classe Simulazione. Aggiunti tutti gli ordini alla coda, il metodo avvia la simulazione richiamando il metodo *simula()* della classe Simulazione.

```
for(int i=0; i<this.ATP.length; i++) {  
    if(Math.random()>=1-probabilita) {  
        Random r = new Random();  
        int qty = r.nextInt((max-min)) + min;  
        Ordine ordine = new Ordine(i, qty);  
        sim.addOrdine(ordine);  
    }  
}  
  
SimulationResult sr = sim.simula();
```

simulaModel()

4.2 Simulazione

La classe è stata creata per ospitare il fulcro dell'algoritmo di simulazione. Gli attributi sono i seguenti.

- Dati di input:
 - Un array di interi con i valori di ATP per ogni mese.
- Dati di output:
 - Una lista di ordini totali ricevuti. Per ogni ordine è possibile sapere il mese in cui è stato richiesto, la quantità di prodotto e il boolean per definire se è accettabile o meno.
 - Il numero degli ordini accettati e di quelli rifiutati.
 - La percentuale degli ordini accettati e di quelli rifiutati.
- La coda degli eventi.

La classe presenta due metodi:

- *addOrdine()* - metodo void con cui, passato dal Model un oggetto di tipo Ordine come parametro, questo viene aggiunto alla coda degli eventi.

```
public void addOrdine(Ordine ordine) {  
    queue.add(new Event(EventType.ARRIVA_ORDINE, ordine));  
}
```

- *simula()* - metodo che restituisce i risultati della simulazione. Attraverso un ciclo while, il metodo itera sulla coda degli eventi fino a che questa diventi vuota. Per ogni ordine in arrivo, ne verifica la fattibilità controllando che non renda negativo l'ATP (come spiegato nella sottosezione 2.2): se risulta

accettabile, setta lo stato dell'ordine a true e incrementa il numero di ordini accettati, altrimenti a false, incrementando il numero degli ordini rifiutati.

```
while(!queue.isEmpty()) {
    Event e = queue.poll();

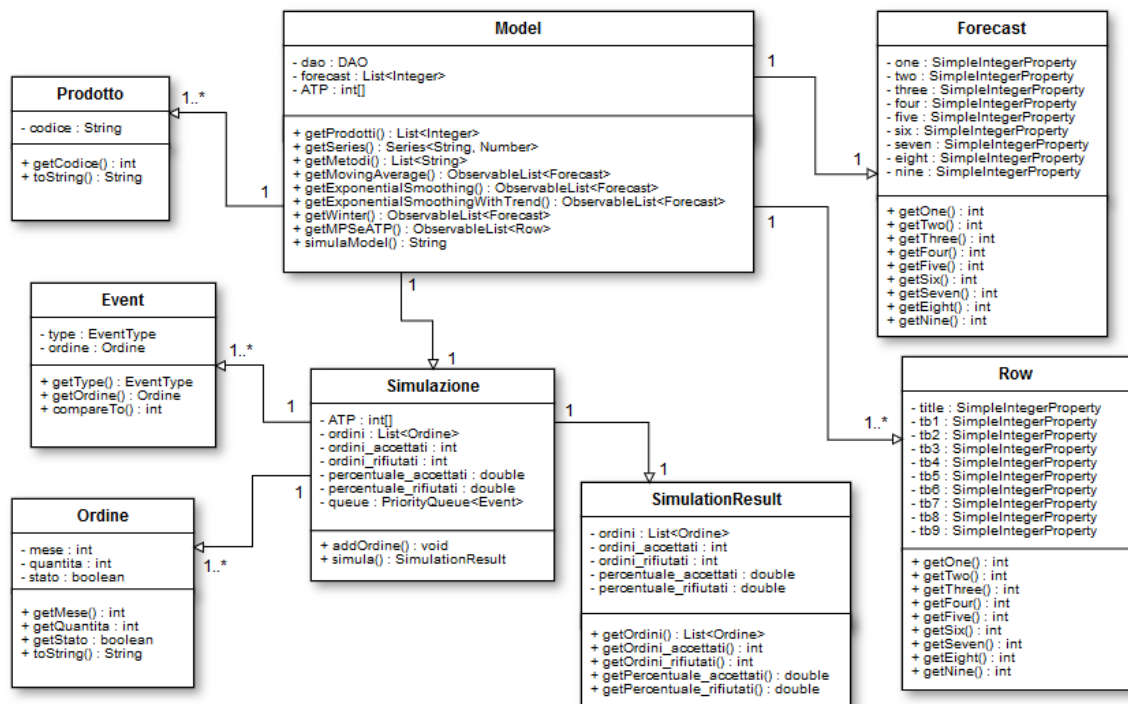
    switch(e.getType()) {
        case ARRIVA_ORDINE :

            for(int i=e.getOrdine().getMese(); i>=0; i--) {
                if(ATP[i]>0) {
                    if(ATP[i]>=e.getOrdine().getQuantita()) {
                        ATP[i] -= e.getOrdine().getQuantita();
                        e.getOrdine().setStato(true);
                        ordini_accettati++;
                        break;
                    }
                    else {
                        ordini_rifiutati++;
                        break;
                    }
                }
            }

            ordini.add(e.getOrdine());
            break;
    }
}
```

5 Diagramma delle classi principali

Di seguito il diagramma delle classi del package tesi.model, in cui sono implementati i metodi più legati alla parte algoritmica.



6 Esempio di svolgimento

Di seguito la spiegazione dei passaggi principali per il funzionamento dell'applicazione.

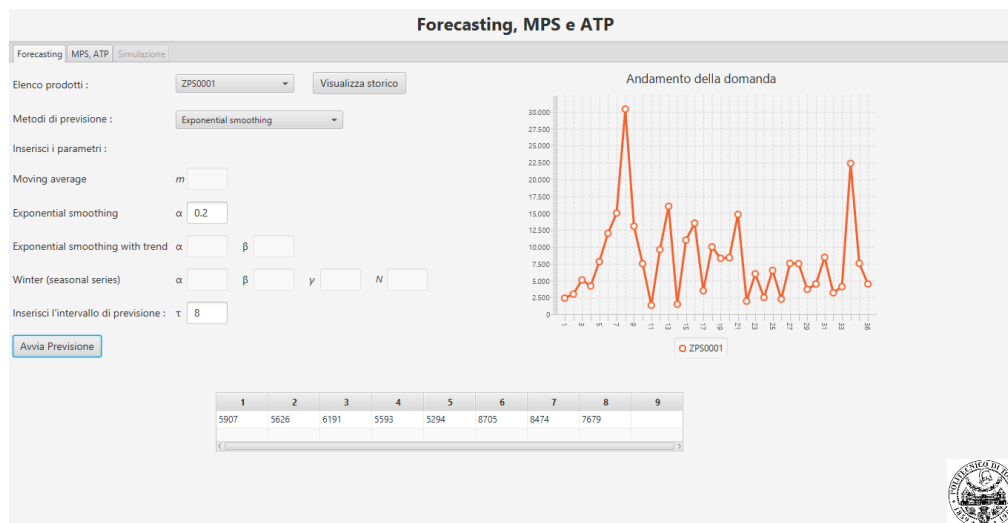
6.1 Esempio : forecasting

Per questa prima fase, l'utente deve scegliere un prodotto tra i quattro disponibili nel data-set di prova, e di questo può scegliere di visualizzare l'andamento della domanda degli ultimi anni attraverso un grafico xy . Per avere intervalli di tempo regolari e procedere con i calcoli, lo storico è stato suddiviso in mesi. E' necessario, poi, che l'utente scelga un metodo di previsione opportuno e ne dia i parametri caratteristici (vedi sottosezione 2.1), e che inserisca un valore di τ (compreso tra 1 e 9 inclusi) per indicare quanti intervalli di tempo vuole prevedere.

Specificati tutti i dati, si può procedere al calcolo.

La scelta del metodo sbagliato, o dei suoi parametri, comporta previsioni prive di significato e non attendibili. E' importante, infatti, che l'utente conosca i prodotti su cui vuole procedere con il calcolo e faccia le dovute assunzioni.

Nell'esempio di seguito, la domanda si mantiene entro un certo range per la maggior parte degli intervalli, ad eccezione di alcuni picchi. L'Exponential Smoothing pesa i valori dello storico dando maggior importanza a quelli più recenti.



6.2 Esempio : MPS e ATP

Effettuata la previsione, è possibile accedere alla parte riguardante MPS e ATP. In questa fase è necessario che l'utente inserisca il proprio valore di MPSLot-size, cioè la quantità di prodotto che verrà ordinata quando l'algoritmo lo richiede, e la disponibilità iniziale del magazzino.

Inoltre, l'utente deve inserire gli ordini acquisiti per i τ mesi specificati in precedenza.

Di seguito sono mostrati due esempi.

Nel primo (*esempio 1*) l'MPSLot-size inserito è troppo per le quantità di ordini previsti e acquisiti. Infatti, il magazzino diviene negativo dopo qualche mese e così anche l'ATP del primo periodo, indici che il piano non funziona.

Nel secondo (*esempio 2*), invece, è stata inserita una quantità più appropriata. Il magazzino, infatti, rimane strettamente positivo per tutti i time bucket e l'ATP riacquista significato. L'MPSLot-size va quindi calibrato in base alla domanda prevista e a quella effettivamente ricevuta, ma anche in base ai costi di stoccaggio e di ordinazione. Starà all'utente, infatti, decidere se per il proprio caso convenga più ordinare spesso o stoccare in magazzino.

Forecasting, MPS e ATP

Forecasting | MPS, ATP | Simulazione

Inserisci l'MPS Lot-size :

Inserisci la quantità iniziale del magazzino :


Prosegui

Inserisci gli ordini acquisiti per i prossimi mesi :

	1	2	3	4	5	6	7	8	9
	5278	5789	5982	5734	5322	7544	8780	7512	

Calcola MPS e ATP

	1	2	3	4	5	6	7	8	9
Previsione	5907	5626	6191	5593	5294	8705	8474	7679	
Ordini acquisiti	5278	5789	5982	5734	5322	7544	8780	7512	
Disponibilità magazzino	4763	3974	2783	2049	1727	-1978	-5758	-8437	
Quantità MPS	5000	5000	5000	5000	5000	5000	5000	5000	
ATP	-3759	0	0	0	0	0	0	0	



esempio 1

Forecasting, MPS e ATP

Forecasting | MPS, ATP | Simulazione

Inserisci l'MPS Lot-size :

Inserisci la quantità iniziale del magazzino :


Prosegui

Inserisci gli ordini acquisiti per i prossimi mesi :

	1	2	3	4	5	6	7	8	9
	5278	5789	5982	5734	5322	7944	8780	7512	

Calcola MPS e ATP

	1	2	3	4	5	6	7	8	9
Previsione	5907	5626	6191	5593	5294	8705	8474	7679	
Ordini acquisiti	5278	5789	5982	5734	5322	7944	8780	7512	
Disponibilità magazzino	14763	8974	2783	12049	6727	13022	4242	11563	
Quantità MPS	15000	0	0	15000	0	15000	0	15000	
ATP	3621	0	0	2220	0	0	0	7488	



esempio 2

6.3 Esempio : simulazione

Calcolati il *Master Production Schedule* e l'*Available to Promise*, viene consentito l'accesso alla fase di simulazione. In questa terza e ultima fase, l'utente deve scegliere un valore di probabilità di ricevere ordini aggiuntivi da attribuire ad ogni time bucket. Dovrà anche stimare un range di valori entro il quale stima si collochi l'eventuale quantità extra per ogni intervallo.

Di seguito un esempio in cui è stata scelta una probabilità del 30% e un range 1000-3500pz.

Forecasting, MPS e ATP

Forecasting | MPS, ATP | Simulazione

Scegli un valore di probabilità da attribuire all'arrivo di ordini extra in un mese :

0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1

Inserisci un range di valori della domanda su cui effettuare la simulazione :

1000 3500

Avvia simulazione

Mese 3, Quantità 1476, Accettato
Mese 6, Quantità 3038, Rifiutato
Mese 8, Quantità 1529, Accettato
Numero ordini accettati : 2
Numero ordini rifiutati : 1
Percentuale ordini accettati : 66.0 %
Percentuale ordini rifiutati : 33.0 %

Mese 1, Quantità 1383, Accettato
Mese 5, Quantità 2004, Accettato

Inizia nuova analisi

Al variare del valore di probabilità scelto, si avranno più o meno ordini aggiuntivi. L'accettazione o meno di questi, dipende dall'ATP, il quale è strettamente legato all'MPS. Una politica per cui si preferisce ordinare spesso in minore quantità (quantità MPSLot-size tendenzialmente bassa) e stoccare quindi poco, risulta generalmente meno flessibile a ordini extra. Anche in questo caso, quindi, vanno valutati i costi di stoccaggio e ordinazione in modo tale che ogni strategia sia conforme alle caratteristiche e alle esigenze dell'azienda.

Per iniziare una nuova analisi, sarà sufficiente cliccare sul bottone "Inizia nuova analisi" e riprendere dalla fase di *forecasting*.

7 Conclusioni

L'applicazione è da considerarsi un aiuto ad un utente che si accinga allo studio della gestione della domanda di un prodotto e al suo piano principale di produzione. Si adatta facilmente a diversi tipi di data-set ed è semplice da capire e usare. E' una versione primaria, che rispecchia conoscenze e competenze attualmente acquisite dall'autore, quindi ancora semplificata in alcuni punti.

L'idea alla base è quella di poter coniugare conoscenze produttive e competenze basilari di programmazione, problematiche gestionali e strumenti informatici, in modo che questi ultimi rappresentino un metodo di risoluzione aggiuntivo e alternativo a problemi aziendali e non solo.

Il codice del progetto è disponibile su GitHub:

Clicca qui

Una demo dell'applicazione è disponibile su YouTube:

Clicca qui