

## Solution des exercices en java

'Polynome.java':

```

public class Polynome {
    private double[] coefficients;
    private int degre;

    public Polynome(int degre) throws ArgumentInvalideException{
        if (degre < 0){
            throw new ArgumentInvalideException("Degre invalide");
        }
        this.degre = degre;
        coefficients = new double[degre+1];
        for (int i = 0; i < degre; i++)
            coefficients[i] = 0;
        coefficients[degre] = 1;
    }

    public void setCoefficient(int degre, double coefficient) throws
ArgumentInvalideException{
        if (degre < 0 || degre > this.degre){
            throw new ArgumentInvalideException("Degre invalide");
        }
        if (coefficient == 0 && degre == this.degre && this.degre !=0){
            throw new ArgumentInvalideException("Degre nul, non autorise pour ce
degre");
        }
        coefficients[degre] = coefficient;
    }

    public double evaluateEn(double x){
        double resultat = 0;
        double tmp = 1;
        for (int i = 0; i <= degre; i++){
            resultat += coefficients[i]*tmp;
            tmp *= x;
        }
        return resultat;
    }

    public String toString(){
        if (degre == 0)
            return "" + this.coefficients[0];
        String result = "" + this.coefficients[degre]+" x^" + degre;
        for (int i = degre-1 ; i > 0 ; i--){
            if (this.coefficients[i] < 0) {
                result += "-" + Math.abs(this.coefficients[i]) + " x^" + i;
            }
            else {
                if (this.coefficients[i] > 0){
                    result += " + " + this.coefficients[i] + " x^" + i;
                }
            }
        }
        if (this.coefficients[0] != 0) result += " " + this.coefficients[0] ;
        return result;
    }

    public double racineParBissection(double a, double b, int decimale) throws
NumeriqueException, MaximumIterateurAtteintException{
        if (a > b){
            double tmp = a;

```

```

        a = b;
        b = tmp;
    }
    double stop = 0.5*Math.pow(10,-decimale);
    double fa = evaluateEn(a);
    if (fa == 0)
        return a;
    double fb = evaluateEn(b);
    if (fb == 0)
        return b;
    if (fa*fb > 0)
    {
        throw new NumeriqueException("Bolzano invalide");
    }
    double m = 0;
    for (int i = 0; i <= 1000; i++){

        double longueur = b-a;
        m = (a+b)/2;
        if (longueur < stop)
            return m;
        double fm = evaluateEn(m);
        if (fa*fm == 0)
            return m;
        if (fa*fm < 0)
            b = m;
        else{
            a = m;
            fa = fm;
        }
    }
    throw new MaximumIterateurAtteintException(m, 1000, "racine Par Bissection
non trouve");
}

public Polynome polynomeDerive(){
    try{
        Polynome derive;
        if (degre == 0){
            derive = new Polynome(0);
            derive.coefficients[0] = 0;
            return derive;
        }
        derive = new Polynome(degre-1);
        for (int i = 0; i < degre; i++)
            derive.coefficients[i] = (coefficients[i+1]*(i+1));
        return derive;
    }
    catch (ArgumentInvalideException e) {
        throw new InternalError();
    }
}

public double racineParLaCorde(double a, double b, int decimale) throws
NumeriqueException{
    return 0.0;
    //To DO
}

public double racineParNewtonAncienne(double a, double b, int decimale) throws
NumeriqueException, MaximumIterateurAtteintException{
    double fa = this.evaluateEn(a);
    double fb = this.evaluateEn(b);
    if (fa == 0) return a;
    if (fb == 0) return b;
    if (fa * fb > 0) throw new NumeriqueException("Bolzano neest pas satisfait");
    double limite = 0.5*Math.pow(10, -decimale);

```

```

Polynome deriveSeconde = this.polynomeDerive().polynomeDerive();
double derive2A = deriveSeconde.evaluateEn(a);
double variable, fixe, fVariable;
if (fa * derive2A < 0){
    variable = b;
    fixe = a;
    fVariable = fb;
} else {
    variable = a;
    fixe = b;
    fVariable = fa;
}
if (polynomeDerive().evaluateEn(variable) * polynomeDerive().evaluateEn(fixe) <
0) throw new NumeriqueException("Bt");
double derive = polynomeDerive().evaluateEn(fixe);
int i = 0;
double variableOld;
while(Math.abs(fVariable/derive)>limite && i < 1000){
    variableOld = variable;
    variable = variable-(fVariable/polynomeDerive().evaluateEn(variable));
    fVariable = this.evaluateEn(variable);
    i++;
    if (Math.abs(variableOld) < Math.abs(variable)) throw new
NumeriqueException("CA BUGGGGGGGGGG");
}
if (i>=1000)throw new MaximumIterateurAtteintException(variable, 1000, "Pas
de solution");
return variable;
}

public double racineParNewton(double a, double b, int decimale) throws
NumeriqueException, MaximumIterateurAtteintException{
    double fa = this.evaluateEn(a);
    double fb = this.evaluateEn(b);
    if (fa == 0) return a;
    if (fb == 0) return b;
    if (fa * fb > 0) throw new NumeriqueException("Bolzano neest pas satisfait");
    double limite = 0.5*Math.pow(10, -decimale);
    Polynome deriveSeconde = this.polynomeDerive().polynomeDerive();
    double derive2A = deriveSeconde.evaluateEn(a);
    double variable, fixe, fVariable;
    if (fa * derive2A < 0){
        variable = b;
        fixe = a;
        fVariable = fb;
    } else {
        variable = a;
        fixe = b;
        fVariable = fa;
    }
    if (polynomeDerive().evaluateEn(variable) * polynomeDerive().evaluateEn(fixe) <
0) throw new NumeriqueException("Bt");
    double derive = polynomeDerive().evaluateEn(fixe);
    int i = 0;
    double variableOld;
    double tropfort = polynomeDerive().evaluateEn(variable);
    while(Math.abs(fVariable/derive)>limite && i < 10000){
        variableOld = variable;
        variable = variable-(fVariable/tropfort);
        fVariable = this.evaluateEn(variable);
        i++;
        if (Math.abs(variableOld) < Math.abs(variable)) throw new
NumeriqueException("CA BUGGGGGGGGGG");
    }
    if (i>=10000)throw new MaximumIterateurAtteintException(variable, 1000, "Pas
de solution");
    return variable;
}

```

```

}

public int getDegre() {
    return degre;
}
}

```

### 'MatriceCarree.java':

```

public class MatriceCarree {

    //Ajoutez le ou les attributs que vous jugerez necessaire
    private double[][] matriceCarree;

    /* Ce constructeur permet de construire la matrice carree nulle de taille n*/
    public MatriceCarree(int taille) throws ArgumentInvalideException {
        if (taille <= 0)
            throw new ArgumentInvalideException ("Taille invalide");
        matriceCarree = new double[taille][taille];
        for (int i = 0; i < taille; i++)
            for (int j = 0; j < taille; j++)
                matriceCarree[i][j] = 0;
    }

    /* Ce constructeur permet de construire la matrice carree correspondant au tableau
    passe en parametre.
    Si le tableau passe en parametre ne correspond pas e une matrice carree, le
    constructeur lance une ArgumentInvalideException*/
    public MatriceCarree(double[][] matrice) throws ArgumentInvalideException {
        if (matrice == null)
            throw new ArgumentInvalideException ("Matrice invalide");
        if (matrice.length == 0 || matrice[0].length == 0)
            throw new ArgumentInvalideException ("Taille invalide");
        for (int i = 0; i < matrice.length; i++)
            if (matrice[i].length != matrice.length)
                throw new ArgumentInvalideException ("Matrice non carree");
        int taille = matrice.length;
        this.matriceCarree = new double[taille][taille];
        for (int i = 0; i < taille; i++)
            for (int j = 0; j < taille; j++)
                this.matriceCarree[i][j] = matrice[i][j];
    }

    /* Ce constructeur permet de construire une copie de la matrice carree passee en
    parametre*/
    public MatriceCarree(MatriceCarree matrice) throws ArgumentInvalideException {
        if (matrice == null)
            throw new ArgumentInvalideException ("Matrice invalide");
        for (int i = 0; i < matrice.getTaille() ; i++)
            if (matrice.matriceCarree[i].length != matrice.matriceCarree.length)
                throw new ArgumentInvalideException ("Matrice non carree");
        int taille = matrice.getTaille();
        this.matriceCarree = new double[taille][taille];
        for (int i = 0; i < taille; i++)
            for (int j = 0; j < taille; j++)
                this.matriceCarree[i][j] = matrice.matriceCarree[i][j];
    }

    /*Cette methode renvoie la taille de la matrice carree*/
    public int getTaille() {
        return matriceCarree.length;
    }

    /*Cette methode renvoie l'element de la ligne i et de la colonne j de la matrice.
    Si cet element n'existe pas, la methode lance une ArgumentInvalideException*/

```

```

public double getElement(int numLigne, int numColonne)
throws ArgumentInvalideException {
    if (numLigne<= 0 || numLigne>matriceCarre.length)
        throw new ArgumentInvalideException("Argument invalide");
    if (numColonne<= 0 || numColonne>matriceCarre.length)
        throw new ArgumentInvalideException("Argument invalide");
    return matriceCarre[numLigne-1][numColonne-1];
}

/*Cette methode remplace l'element de la ligne i et de la colonne j de la matrice
par l'element passe en parametre.
Si l'element de position (i,j) n'existe pas, la methode lance une
ArgumentInvalideException*/
public void setElement(int numLigne, int numColonne, double element)
throws ArgumentInvalideException {
    if (numLigne<= 0 || numLigne>matriceCarre.length)
        throw new ArgumentInvalideException("Argument invalide");
    if (numColonne<= 0 || numColonne>matriceCarre.length)
        throw new ArgumentInvalideException("Argument invalide");
    matriceCarre[numLigne-1][numColonne-1]=element;
}

/*Cette methode remplace, si possible, la colonne correspondant e numColonne par le
vecteur passe en parametre.
En cas de probleme, elle lancera une ArgumentInvalideException*/
public Vecteur setColonne(int numColonne, Vecteur vecteur)
throws ArgumentInvalideException {
    if (vecteur == null)
        throw new ArgumentInvalideException("Argument invalide");
    if (matriceCarre.length != vecteur.getTaille())
        throw new ArgumentInvalideException("Argument invalide");
    if (numColonne<= 0 || numColonne>matriceCarre.length)
        throw new ArgumentInvalideException("Argument invalide");
    int taille=vecteur.getTaille();
    double elt[] = new double [taille];
    for (int i=0; i<taille; i++){
        elt[i] = matriceCarre[i][numColonne-1];
        matriceCarre[i][numColonne-1]=vecteur.getElement(i+1);
    }
    Vecteur v = new Vecteur(elt);
    return v;
}

/*Cette methode renvoie la valeur du determinant de la matrice*/
public double determinant() {
    if (matriceCarre.length== 1)
        return matriceCarre[0][0];
    double result = 0;
    MatriceCarree matriceATravailler;
    int ligne = 1;
    for (int i = 1; i <= matriceCarre.length; i++){
        matriceATravailler = couperMatrice(ligne,i);
        double toto = matriceATravailler.determinant();
        if (i%2 == 0)
            result -= this.matriceCarre[0][i-1]*toto;
        else
            result += this.matriceCarre[0][i-1]*toto;
    }
    return result;
}

private MatriceCarree couperMatrice(int numLigne, int numColonne) {
    MatriceCarree aRenvoyer;
    try {
        aRenvoyer = new MatriceCarree(matriceCarre.length-1);
    } catch (ArgumentInvalideException e) {

```

```

        throw new InternalError();
    }
    for (int i = 0; i < numLigne-1; i++){
        for(int j = 0; j < numColonne-1; j++)
            aRenvoyer.matriceCarre[i][j] = this.matriceCarre[i][j];
        for(int j = numColonne; j < matriceCarre.length; j++)
            aRenvoyer.matriceCarre[i][j-1] = this.matriceCarre[i][j];
    }
    for (int i = numLigne; i < matriceCarre.length; i++){
        for(int j = 0; j < numColonne-1; j++)
            aRenvoyer.matriceCarre[i-1][j] = this.matriceCarre[i][j];
        for(int j = numColonne; j < matriceCarre.length; j++)
            aRenvoyer.matriceCarre[i-1][j-1] = this.matriceCarre[i][j];
    }
    return aRenvoyer;
}
public boolean estTriangulaireSuperieure(){
    int taille = matriceCarre.length;
    for (int ligne=0; ligne<taille; ligne++){
        for (int collone=0; collone<ligne; collone++)
            if(matriceCarre [ligne][collone] != 0) return false;
    }
    return true;
}
public Vecteur produit(Vecteur unVecteur) throws ArgumentInvalideException {
    if(matriceCarre.length != unVecteur.getTaille() || unVecteur == null) throw
new ArgumentInvalideException("pas mm taille ou null");
    Vecteur arenvoyer = new Vecteur(matriceCarre.length);
    for(int i=0; i<matriceCarre.length; i++){
        double total = 0;
        for(int y=0; y<matriceCarre.length; y++){
            total += matriceCarre[i][y]*unVecteur.getElement(y+1);
        }
        arenvoyer.setElement(i+1, total);
    }
    return arenvoyer;
}

public void permuter(int i,int j) throws ArgumentInvalideException{
    int taille = matriceCarre.length;
    int ligneP = i-1;
    int ligneD = j-1;
    if(i<=0 || i>taille || j<=0 || j>taille){
        throw new ArgumentInvalideException("Argument invalide exception");
    }
    double[] temp = matriceCarre [ligneP];
    matriceCarre [ligneP] =matriceCarre [ligneD];
    matriceCarre [ligneD] = temp;
}

public String toString() {
    String ch = "";
    for (int i=0; i< matriceCarre.length; i++){
        for (int j=0; j< matriceCarre.length; j++){
            ch += matriceCarre[i][j]+" ";
        }
        ch += "\n";
    }
    return ch;
}
}

```

'Vecteur.java':

```
public class Vecteur {
```

```
//Ajoutez le ou les attributs que vous jugerez necessaire.
```

```

double[] vecteur;
int taille;

/*ce constructeur permet de construire le vecteur nul de taille indiquee*/
public Vecteur(int taille) throws ArgumentInvalideException {
    if (taille <= 0)
        throw new ArgumentInvalideException ("Taille invalide");
    this.taille = taille;
    vecteur = new double[taille];
    for (int i = 0; i < taille; i++)
        vecteur[i] = 0;
}

/*Ce constructeur permet de construire le vecteur correspondant au tableau passe en
parametre.
Si le tableau passe en parametre ne correspond pas e un vecteur, le constructeur
lance une ArgumentInvalideException.*/

public Vecteur(double[] elt) throws ArgumentInvalideException{
    if (elt == null)
        throw new ArgumentInvalideException ("Vecteur invalide");
    if (elt.length == 0)
        throw new ArgumentInvalideException ("Taille invalide");
    this.taille = elt.length;
    vecteur = new double[taille];
    for (int i = 0; i < taille; i++)
        vecteur[i] = elt[i];
}

/*Ce constructeur construit une copie du vecteur passe en parametre*/
public Vecteur(Vecteur v) throws ArgumentInvalideException{
    if (v == null)
        throw new ArgumentInvalideException ("Vecteur invalide");
    if (v.taille == 0)
        throw new ArgumentInvalideException ("Taille invalide");
    taille = v.taille;
    vecteur = new double[taille];
    for (int i = 0; i < taille; i++)
        vecteur[i] = v.vecteur[i];
}

/* Cette methode renvoie l'element de position i du vecteur s'il existe*/
public double getElement(int i) throws ArgumentInvalideException{
    if (i > taille)
        throw new ArgumentInvalideException ("Position invalide");
    if (i <= 0)
        throw new ArgumentInvalideException ("Position invalide");
    return vecteur[i-1];
}

/*Cette methode remplace l'element de position i du vecteur par l'element passe en
parametre*/
public void setElement(int i, double element) throws ArgumentInvalideException{
    if (i > taille)
        throw new ArgumentInvalideException ("Position invalide");
    if (i <= 0)
        throw new ArgumentInvalideException ("Position invalide");
    vecteur[i-1] = element;
}

/*Cette methode renvoie la taille du vecteur*/
public int getTaille() {
    return taille;
}

public Vecteur somme(Vecteur v) throws ArgumentInvalideException {
    if(this.taille != v.getTaille() || v == null) throw new
ArgumentInvalideException("pas mm taille ou null");
}

```

```

    Vecteur arenvoyer = new Vecteur(taille);
    for (int i=1; i<taille+1; i++){
        arenvoyer.setElement(i, this.vecteur[i-1]+v.getElement(i));
    }
    return arenvoyer;
}

    public boolean equivalentAMDecimale(Vecteur v, int m) throws
ArgumentInvalideException {
    if(this.taille != v.getTaille() || v == null) throw new
ArgumentInvalideException("pas mm taille ou null");
    double exposant = Math.pow(10,m);
    for (int i=1; i<taille+1; i++){
        double nb1 = this.vecteur[i-1]*exposant;
        double nb2 = v.getElement(i)*exposant;
        if(Math.round(nb1)/exposant != Math.round(nb2)/exposant) return false;
    }
    return true;
}

    public String toString(){
    String resultat = "";
    for (int i = 0; i < taille; i++)
        resultat += vecteur[i] + "\t";
    return resultat;
}
}
}

```

'systèmeDEquation.java':

```

public class SystemeDEquations {
    MatriceCarree matrice;
    Vecteur vecteur;

    public SystemeDEquations (int n) {
        try {
            matrice = new MatriceCarree(n);
            for (int i = 0; i < n; i++)
                setCoefficient(i+1, i+1, 1.0);
            vecteur = new Vecteur(n);
        } catch (ArgumentInvalideException e) {
            throw new InternalError();
        }
    }

    private SystemeDEquations (SystemeDEquations sys){
        try {
            matrice = new MatriceCarree(sys.matrice);
            vecteur = new Vecteur(sys.vecteur);
        } catch (ArgumentInvalideException e) {
            throw new InternalError();
        }
    }

    public void setCoefficient(int i, int j, double nvCoef) throws
ArgumentInvalideException{
        matrice.setElement(i, j, nvCoef);
    }

    public void setTermeConstant (int i, double nvTermeConst) throws
ArgumentInvalideException{
        vecteur.setElement(i, nvTermeConst);
    }
}

```

```

public String toString(){
    return matrice.toString()+"\n"+vecteur.toString();
}

public Vecteur solutionParCramer() throws NumeriqueException{
    try {
        Vecteur solution = new Vecteur(vecteur.getTaille());
        double delta = matrice.determinant();
        if (delta == 0)
            throw new NumeriqueException("Math error : Le delta ne peut etre
egal e zero");
        for (int i=1; i<=matrice.getTaille(); i++){
            Vecteur colonneTmp = matrice.setColonne(i, vecteur);
            double deltaBis = matrice.determinant();
            solution.setElement(i, (deltaBis/delta));
            matrice.setColonne(i, colonneTmp);
        }
        return solution;
    } catch (ArgumentInvalideException e) {
    }
    return null;
}

private Vecteur solutionSystemeTriangulaire()throws NumeriqueException{
    Vecteur x = null;
    try {
        for (int i=1; i<=matrice.getTaille(); i++){
            if(matrice.getElement(i,i)==0)
                throw new NumeriqueException("Math error ");
        }
        x = new Vecteur(matrice.getTaille());
        double sol;
        for (int i=matrice.getTaille(); i>=1; i--){
            sol=0;
            for (int k=i+1; k<=matrice.getTaille(); k++){
                sol += matrice.getElement(i, k)*x.getElement(k);
            }
            x.setElement(i, 1/matrice.getElement(i,
i)*(this.vecteur.getElement(i)-sol));
        }
    } catch (ArgumentInvalideException e) {
        throw new InternalError();
    }
    return x;
}

public Vecteur solutionParGauss() throws NumeriqueException{
    return this.systemeTriangulaire().solutionSystemeTriangulaire();
}

public Vecteur calculeBeta() throws ArgumentInvalideException {
    int taille = vecteur.getTaille();
    Vecteur arenvoyer = new Vecteur(taille);
    for (int i= 1; i<=taille; i++){
        double b = vecteur.getElement(i);
        double a = matrice.getElement(i, i);
        arenvoyer.setElement(i, b/a);
    }
    return arenvoyer;
}

private MatriceCarree calculeAlpha() throws ArgumentInvalideException{
    int taille = matrice.getTaille();
    MatriceCarree aRenvoyer = new MatriceCarree(taille);
    for (int i=1; i<=taille; i++) {
        double aii = matrice.getElement(i, i);

```

```

        for (int j=1; j<=i-1; j++) {
            double aij = matrice.getElement(i, j);
            aRenvoyer.setElement(i, j, -(aij/aii));
        }
        for (int j=i+1; j<=taille; j++) {
            double aij = matrice.getElement(i, j);
            aRenvoyer.setElement(i, j, -(aij/aii));
        }
        aRenvoyer.setElement(i, i, 0);
    }
    return aRenvoyer;
}
private boolean verificationConvergence(){
    try{
        int taille = matrice.getTaille();
        for(int i=1; i<=taille; i++){
            double totalAlpha = 0;
            for (int j=1; j<=i-1; j++){
                totalAlpha += Math.abs(matrice.getElement(i, j));
            }
            for (int j=i+1; j<=taille; j++){
                totalAlpha += Math.abs(matrice.getElement(i, j));
            }
            if(Math.abs(matrice.getElement(i, i))<=totalAlpha) return false;
        }
    } catch (ArgumentInvalideException e) {
        throw new InternalError();
    }
    return true;
}
public Vecteur solutionParApproximation( int NombreDeDecimalesexactes) throws
NumeriqueException{
    if(!verificationConvergence())throw new NumeriqueException("La verification
de convergence n'a pas r ussit");
    try{
        Vecteur beta = calculeBeta();
        MatriceCarree aplha = calculeAlpha();
        Vecteur solutionApproxim = calculeBeta();
        Vecteur solutionApproximPred;
        do{
            solutionApproximPred = solutionApproxim;
            solutionApproxim =
beta.somme(aplha.produit(solutionApproximPred));
        }while(!
solutionApproxim.equivalentAMDecimale(solutionApproximPred,NombreDeDecimalesexactes));
        return arrondir(solutionApproxim,NombreDeDecimalesexactes+1);
    } catch (ArgumentInvalideException e) {
        throw new InternalError();
    }
}
public Vecteur solutionParSeidel( int NombreDeDecimalesexactes) throws
NumeriqueException{
    if(!verificationConvergence())throw new NumeriqueException("La verification
de convergence n'a pas r ussit");
    try{
        Vecteur beta = calculeBeta();
        MatriceCarree alpha = calculeAlpha();
        Vecteur solutionApproxim = new Vecteur(beta);
        Vecteur solutionApproximPred;
        do{
            solutionApproximPred = new Vecteur(solutionApproxim);
            for(int i=1; i<=alpha.getTaille(); i++){
                double addAlpha = 0;
                for(int y=1; y<=alpha.getTaille(); y++){
                    addAlpha +=
alpha.getElement(i,y)*solutionApproximPred.getElement(y);
                }
            }
        }
    }
}

```

```

        solutionApproxim.setElement(i, addAlpha+beta.getElement(i));
    }
    }while(!
solutionApproxim.equivalentAMDecimale(solutionApproximPred,NombreDeDecimalesexactes));
    return arrondir(solutionApproxim,NombreDeDecimalesexactes+1);
} catch (ArgumentInvalideException e) {
    throw new InternalError();
}
}
private Vecteur arrondir(Vecteur v, int nbDec){
    double mul = Math.pow(10, nbDec);
    for (int i=1; i<=v.getTaille(); i++){
        try {
            v.setElement(i, Math.round(v.getElement(i)*mul)/mul);
        } catch (ArgumentInvalideException e) {
            throw new InternalError();
        }
    }
    return v;
}
private SystemeDEquations systemeTriangulaire() {
    SystemeDEquations solution = new SystemeDEquations(this);
    try {
        for (int r=1; r<=solution.matrice.getTaille(); r++){
            solution.pivotage(r);
            double arr = solution.matrice.getElement(r,r);
            for (int i=r+1; i<=solution.matrice.getTaille(); i++){
                if(arr == 0) break;
                double m = solution.matrice.getElement(i,r)/arr;
                solution.matrice.setElement(i, r, 0);
                for(int colI=r+1; colI<=solution.matrice.getTaille();colI++)
{
                    double element = solution.matrice.getElement(i,
colI)-(m*solution.matrice.getElement(r,colI));
                    solution.matrice.setElement(i, colI, element);
                }
                double element = (solution.vecteur.getElement(i) -
(m*solution.vecteur.getElement(r)));
                solution.vecteur.setElement(i, element);
            }
        }
    } catch (ArgumentInvalideException e) {
        throw new InternalError();
    }
    return solution;
}
private void pivotage(int numElement){
    try {
        double elementMAX = Math.abs(matrice.getElement(numElement,
numElement));
        int numLigneAPivoter = numElement;
        for (int ligne = numElement+1; ligne <= matrice.getTaille(); ligne++){
            double nvElement = Math.abs(matrice.getElement(ligne,
numElement));

            if (nvElement > elementMAX){
                elementMAX = nvElement;
                numLigneAPivoter = ligne;
            }
        }
        if (numLigneAPivoter != numElement){
            matrice.permuter(numElement, numLigneAPivoter);
            double elementTemp = vecteur.getElement(numElement);
            vecteur.setElement(numElement,
vecteur.getElement(numLigneAPivoter));
            vecteur.setElement(numLigneAPivoter, elementTemp);
        }
    } catch (ArgumentInvalideException e1) {

```

