**Приложение А**

**Текст программы**

> restart:

> with(inttrans):

> with(LinearAlgebra):

Normalizer := simplify:

**Входные параметры**

> k:=0.1:

> B := Vector[row]([parse(DocumentTools:-GetProperty('B\_coeff','value'))]):

> A := Vector[row]([parse(DocumentTools:-GetProperty('A\_coeff','value'))]):

> phi\_z := parse(DocumentTools:-GetProperty('phi\_z\_', 'value')):

> L\_z := parse(DocumentTools:-GetProperty('L\_z\_','value')):

> t\_u := 0.5: # Желаемое время установления

> c\_0\_желаемое := 0:

> c\_1\_желаемое := parse(DocumentTools:-GetProperty('с\_1\_','value')):

> phi\_gr := parse(DocumentTools:-GetProperty('phi\_gr\_','value')):

> omega\_z\_beg := parse(DocumentTools:-GetProperty('omega\_z\_beg\_','value')):

> omega\_z\_end := parse(DocumentTools:-GetProperty('omega\_z\_end\_','value')):

> numberOfSteps := parse(DocumentTools:-GetProperty('numberOfSteps\_','value')):

**Критерий Гурвица**

> GurvitzMatrix := proc (C::Vector)

local i::posint, j::posint,M::Matrix,n::posint,curIndex::posint;

n := Dimension(C)-1; # Степень полинома

M := Matrix(n,n);

for i from 1 to n do

for j from 1 to n do

curIndex := n - j\*2 + i;

if (curIndex < 0 or curIndex > n) then

M[i,j]:=0;

else

M[i,j]:=C[curIndex+1];

end if;

end do;

end do;

M;

end proc:

> GurvitzMinorDeterminants := proc (C::Vector) :: Vector;

local G::Matrix,n::posint,M::Vector,i::posint;

G := GurvitzMatrix(C);

n := RowDimension(G)-1;

M := Vector[row](n+1);

M[1] := G[1,1];

if (n <> 0) then

for i from 1 to n do

M[i+1] := Determinant(SubMatrix(G,[1..i+1],[1..i+1]));

end do;

end if;

return M;

end proc:

> RauseGurvitzCriterion := proc (C::Vector) :: string;

local M::Vector,s::real,i::posint,n::posint;

M := GurvitzMinorDeterminants(C);

n := Dimension(M);

s := sign(C[-1]);

for i from 1 to n do

if (s^i\*M[i] < 0) then return "Полином неустойчив";

elif (i=n-1 and M[i]=0) then return "Колебательная граница устойчивости";

elif (i=n and M[i]=0) then return "Апериодическая граница устойчивости";

end if;

end do;

"Полином устойчив";

end proc:

**Функция нахождения максимального элемента массива**

> findMax := proc(C::Vector) :: float;

local i::posint,n::posint,max;

n := Dimension(C);

max := C[1];

for i from 2 to n do

#printf("max=%f ; C[%d]=%f\n",max,i,C[i]);

if max < C[i] then

max := C[i];

end if;

end do;

#printf("Exiting findMax\n");

max;

end proc:

**Функция нахождения минимального элемента массива**

> findMin := proc(C::Vector) :: float;

local i::posint,n::posint,min::float;

n := Dimension(C);

min := C[1];

for i from 2 to n do

if min > C[i] then min := C[i]; end if;

end do;

min;

end proc:

**Функция генерации полинома по вектору коэффициентов**

> generatePoly := proc(Coefficients::Vector,s) :: polynom;

local polyToReturn::polynom,i::posint;

polyToReturn :=0;

for i from 1 to Dimension(Coefficients) do

polyToReturn := polyToReturn + Coefficients[i]\*s^(i-1);

end do;

polyToReturn;

end proc:

**Функция коррекции фазы**

> CorPhi := proc(W,omega\_v::Vector) :: Vector;

local i::posint,n::posint,phi::Vector;

phi := Vector[row](Dimension(omega\_v));

phi := map(W,I\*omega\_v);

phi := map(argument,phi);

for i from 2 to Dimension(omega\_v) do

n := round((phi[i]-phi[i-1])/(2\*Pi));

if (n <> 0) then

phi[i]:=phi[i]-2\*Pi\*n;

end if;

end do;

phi;

end proc:

**Функция расчета времени установления по переходной характеристике**

h - функция переходной характеристики

W\_0=h(infinity) - значение передаточной функции в нуле

W\_inf=h(0) - значение передаточной функции в пределе при значении частоты стремящейся к бесконечности

T - верхний предел времени установления

> getTransientPeriod2 := proc(h,W\_0,W\_inf,T)

local

dt::float,

numberOfPoints::posint,

t\_k::Vector,

h\_k::Vector,

hv::Vector,

f::function,

j::posint,

Delta::float,

k::posint,

ReEval::function,

max\_h\_k::float,

min\_h\_k::float,

max\_v::float;

ReEval := num->Re(num);

dt := 1;

numberOfPoints:=round(T/dt);

f:= j -> j\*T/numberOfPoints:

t\_k:=Vector[row](numberOfPoints,f):

h\_k:=map(h,t\_k);

h\_k:=map(ReEval,h\_k);

# Отклонение от предельного значения

# при котором считаем переходную характеристику

# установившейся

max\_h\_k := findMax(h\_k);

min\_h\_k := findMin(h\_k);

max\_v := max(abs(W\_inf-max\_h\_k),abs(W\_inf-min\_h\_k));

Delta:=0.05\*abs(max\_v);

for k from 1 to numberOfPoints do

#printf("Begining it No. %d\n",k);

hv := SubVector(h\_k,[k..-1])-Vector[row](1..(Dimension(h\_k)-(k-1)),W\_0);

#printf("%{x}4f\n",hv);

if max(findMax(hv),-findMin(hv))<=Delta then

return t\_k[k];

end if;

end do;

T;#"Нет сходимости";

end proc:

**Функция расчета времени установления по переходной характеристике с проверкой устойчивости**

На вход будем подавать характеристический полином и переходную характеристику.

Полином - для определения устойчивости.

**> getTransientPeriod := proc(C::Vector,h,W\_0,W\_inf)**

**local eta::float,**

**T::float,**

**ReEval::function;**

**if (RauseGurvitzCriterion(C)<>"Полином устойчив")**

**then return "Нет сходимости";**

**end if;**

**# Определяем степень устойчивости**

**printf("before eta\n");**

**ReEval := num->evalf(Re(num));**

**eta:=-findMax(convert(map(ReEval,[solve(generatePoly(C,s))]),Vector));**

**printf("after eta\n");**

**# Определяем верхнюю оценку времени установления**

**T := (3/eta)\*10;**

**return getTransientPeriod2(h,W\_0,W\_inf,T);**

**end proc:**

**Функция расчета перерегулирования по переходной характеристике**

> getTransientOvershoot := proc(h,W\_0,C::Vector)

local

h\_0::float,

h\_max::float,

h\_min::float,

T::float,

eta::float,

ReEval::function;

h\_0 := h(0);

#printf("W\_0=%f\n",W\_0);

#printf("h\_0=%f\n",h\_0);

if (RauseGurvitzCriterion(C)="Полином неустойчив")

then return "Перерегулирование неопределимо ввиду отсутствия установившегося значения переходной характеристики";

end if;

# Определяем степень устойчивости

#printf("before eta\n");

ReEval := num->evalf(Re(num));

eta:=-findMax(convert(map(ReEval,[solve(generatePoly(C,s))]),Vector));

# Определяем верхнюю оценку времени установления

T := (3/eta)\*3;

printf("T:=%f\n",T);

h\_max := Optimization:-Maximize(h(evalf(t)),t=0..T,method=branchandbound)[1];

h\_min := Optimization:-Minimize(h(evalf(t)),t=0..T,method=branchandbound)[1];

printf("h\_max=%f\n",h\_max);

printf("h\_min=%f\n",h\_min);

if (h\_0 < W\_0 and W\_0 < h\_max) then return (h\_max - W\_0)/(W\_0 - h\_0); end if;

if (h\_0 > W\_0 and W\_0 > h\_min) then return (h\_min - W\_0)/(W\_0 - h\_0); end if;

return 0;

end proc:

**Расчет частотных характеристик**

> A\_ := s->generatePoly(A,s);

> B\_ := s->generatePoly(B,s);

> W\_исх := s->(B\_(s))/(A\_(s));

> evalf(W\_исх(omega\*sqrt(-1)));

> j:= I;

> P := omega->(Re(W\_исх(I\*omega)));

> Q := omega->(Im(W\_исх(I\*omega)));

> evalf[3](Q(omega\*I));

> DocumentTools:-SetProperty('P\_',value,plot(P(omega),omega=0..0.5,thickness = 3));

> DocumentTools:- SetProperty('Q\_',value,plot(Q(omega),omega=0..0.5, thickness=3));

> Amp := omega->(sqrt(P(omega)^2 + Q(omega)^2)):

> DocumentTools:-SetProperty('A\_',value,plot(Amp(omega),omega=0..1.5,thickness=3));

> numberOfPoints := 100:

> begOmega:=0.01:

> endOmega:=1:

> f:= (j) -> (begOmega + j\*((endOmega - begOmega)/numberOfPoints)):

omega\_v:=Vector[row](numberOfPoints,f):

> Phi := omega->arctan(Q(omega),P(omega)):

> Phi\_v:= CorPhi(W\_исх,omega\_v):

> f:= (j) -> [omega\_v[j],Phi\_v[j]]:

vectorToPlot:=Vector[row](numberOfPoints,f):

> with(plots):

DocumentTools:-SetProperty('Phi\_',value,pointplot(vectorToPlot,connect=true,color=red,thickness=3));

> L := omega->20\*log10(Amp(omega)):

DocumentTools:-SetProperty('L\_',value,plot(L(omega),omega=0..1.5,thickness=3,axis[1]=[mode=log]));

> with(plots):

DocumentTools:-SetProperty('LPhi\_',value,pointplot(vectorToPlot,connect=true,color=red,thickness=3,axis[1]=[mode=log]));

**Расчет временных характеристик**

> RauseGurvitzCriterion(A);

> h\_original := t->evalf(invlaplace(W\_исх(s)/s,s,t));

> evalf[3](h\_original(t));

> evalf(h\_original(1));

> T\_уст := getTransientPeriod(A,h\_original,W\_исх(0),h\_original(0));

> DocumentTools:-SetProperty('t\_ust\_',value,evalf[4](T\_уст));

> if T\_уст="Нет сходимости" then

T3\_уст:=1000;

else

T3\_уст := T\_уст\*3;

end if;

>

> DocumentTools:-SetProperty('h\_',value,plot([h\_original(t),W\_исх(0)+abs(W\_исх(0)-h\_original(0))\*0.05,W\_исх(0)-abs(W\_исх(0)-h\_original(0))\*0.05],t=0..T3\_уст,color=[red,blue,blue],thickness=[3,1,1]));

> w\_original := t->evalf(invlaplace(W\_исх(s),s,t)):

> DocumentTools:-SetProperty('w\_',value,plot(w\_original(t),t=0..T3\_уст,thickness=3));

**Расчет коэффициентов регулятора**

> K := W\_исх(0):

> K\_i := 1/(K\*c\_1\_желаемое):

> DocumentTools:-SetProperty('K\_i\_',value,evalf[3](K\_i));

> lambda\_z := 10^(0.05\*L\_z);

> K\_p := omega\_z->evalf((cos(phi\_gr+phi\_z-Phi(omega\_z)))/(lambda\_z\*Amp(omega\_z)));

> DocumentTools:-SetProperty('K\_p\_',value,plot(K\_p(omega\_z),omega\_z=omega\_z\_beg..omega\_z\_end,thickness=3,axis[1]=[mode=log]));

> K\_d := omega\_z->evalf(((sin(phi\_gr+phi\_z-Phi(omega\_z)))/(lambda\_z\*Amp(omega\_z)) + K\_i/omega\_z)/omega\_z);

> DocumentTools:-SetProperty('K\_d\_',value,plot(K\_d(omega\_z),omega\_z=omega\_z\_beg..omega\_z\_end,thickness=3,axis[1]=[mode=log]));

> R := (s,omega\_z)->evalf((K\_i + K\_p(omega\_z)\*s + K\_d(omega\_z) \* s^2)/s);

>

> W\_разомкнутая := (s,omega\_z)->W\_исх(s)\*R(s,omega\_z);

> W\_замкнутая := (s,omega\_z)->W\_разомкнутая(s,omega\_z)/(1+W\_разомкнутая(s,omega\_z));

>

> h\_endless := (t,omega\_z)->invlaplace(W\_замкнутая(s,omega\_z)/s,s,t);

**Получение оценок времени установления и перерегулирования конечной системы**

> getUpperBound := proc(A::Vector,B::Vector,K\_i,K\_p,K\_d,omega\_z) :: float;

local ReEval::function, eta::float;

ReEval := num->evalf(Re(evalf(num)));

eta:=-findMax(convert(map(ReEval,[solve(generatePoly(B,s)\*(K\_i+s\*K\_p(omega\_z)+s^2\*K\_d(omega\_z))+s\*generatePoly(A,s))]),Vector));

if eta<=0 then return infinity; end if;

3/eta;

end proc:

> getLowerBound := proc(A::Vector,B::Vector,K\_i,K\_p,K\_d,omega\_z) ::float;

local ReEval::function, gamma::float,eta::float;

ReEval := num->evalf(Re(evalf(num)));

gamma:=-findMin(convert(map(ReEval,[solve(generatePoly(B,s)\*(K\_i+s\*K\_p(omega\_z)+s^2\*K\_d(omega\_z))+s\*generatePoly(A,s))]),Vector));

eta:=-findMax(convert(map(ReEval,[solve(generatePoly(B,s)\*(K\_i+s\*K\_p(omega\_z)+s^2\*K\_d(omega\_z))+s\*generatePoly(A,s))]),Vector));

if eta<=0 then return infinity; end if;

3/gamma;

end proc:

> getOvershoot := proc(A,B,K\_i,K\_p,K\_d,omega\_z) :: float;

local ReEval::function, eta::float,ReImEval::function,mu::float;

ReEval := num->evalf(Re(evalf(num)));

eta:=-findMax(convert(map(ReEval,[solve(generatePoly(B,s)\*(K\_i+s\*K\_p(omega\_z)+s^2\*K\_d(omega\_z))+s\*generatePoly(A,s))]),Vector));

if eta<=0 then return infinity; end if;

ReImEval := num->evalf(abs(Im(num)/Re(num)));

mu := findMax(convert(map(ReImEval,[solve(generatePoly(B,s)\*(K\_i+s\*K\_p(omega\_z)+s^2\*K\_d(omega\_z))+s\*generatePoly(A,s))]),Vector));

exp(-Pi/mu);

end proc:

> findNonZeroMin := proc(C::Vector) :: posint;

local i::posint,n::posint,min::float,min\_i::posint;

n := Dimension(C);

min := 1000000000;

min\_i:=1;

for i from 1 to n do

if min > C[i] and C[i]<>0 then

min := C[i];

min\_i:=i;

end if;

end do;

min\_i;

end proc:

> findIndexOfMin := proc(C::Vector) :: posint;

local i::posint,n::posint,min::float,min\_i::posint;

n := Dimension(C);

min := C[1];

min\_i:=1;

for i from 2 to n do

if min > C[i] then

min := C[i];

min\_i:=i;

end if;

end do;

min\_i;

end proc:

> f:= (j) -> (omega\_z\_beg + j\*((omega\_z\_end - omega\_z\_beg)/numberOfSteps)):

omega\_z\_v:=Vector[row](numberOfSteps,f):

> mapFuncUpperBound := omega\_z->evalf(getUpperBound(A,B,K\_i,K\_p,K\_d,omega\_z)):

> upperBound\_v := map(mapFuncUpperBound,omega\_z\_v):

> f:= (j) -> [omega\_z\_v[j],upperBound\_v[j]]:

vectorToPlot:=Vector[row](numberOfSteps,f):

> with(plots):

up := pointplot(vectorToPlot,connect=true,color=red,thickness=3,axis[1]=[mode=log],legend="Верхняя оценка времени установления"):

>

> f:= (j) -> (omega\_z\_beg + j\*((omega\_z\_end - omega\_z\_beg)/numberOfSteps)):

omega\_z\_v:=Vector[row](numberOfSteps,f):

> mapFuncLowerBound := omega\_z->evalf(getLowerBound(A,B,K\_i,K\_p,K\_d,omega\_z)):

> lowerBound\_v := map(mapFuncLowerBound,omega\_z\_v):

> f:= (j) -> [omega\_z\_v[j],lowerBound\_v[j]]:

vectorToPlot1:=Vector[row](numberOfSteps,f):

> with(plots):

lower:= pointplot(vectorToPlot1,connect=true,color=blue,thickness=3,axis[1]=[mode=log],legend="Нижняя оценка времени установления"):

>

> DocumentTools:-SetProperty('t\_ust\_plot',value,display(up,lower));

> mapFuncOvershoot := omega\_z->evalf(getOvershoot(A,B,K\_i,K\_p,K\_d,omega\_z)):

> overshoot\_v := map(mapFuncOvershoot,omega\_z\_v):

> f:= (j) -> [omega\_z\_v[j],overshoot\_v[j]]:

vectorToPlot1:=Vector[row](numberOfSteps,f):

> with(plots):

overshoot:= pointplot(vectorToPlot1,connect=true,color=blue,thickness=3,axis[1]=[mode=log]):

> DocumentTools:-SetProperty('sigma\_plot',value,display(overshoot)):

>

**Расчет коэффициентов регулятора по критерию минимальной оценки времени установления**

> minOfUpperBound := findNonZeroMin(upperBound\_v):

> DocumentTools:-SetProperty('omega\_max',caption,evalf[3](omega\_z\_v[minOfUpperBound]));

> DocumentTools:-SetProperty('R\_s\_',value,evalf[3](R(s,omega\_z\_v[minOfUpperBound])));

> DocumentTools:-SetProperty('W\_z\_s',value,evalf[3](rationalize(W\_замкнутая(s,omega\_z\_v[minOfUpperBound]))));

> DocumentTools:-SetProperty('t\_max\_',caption,evalf[3](upperBound\_v[minOfUpperBound]));

> DocumentTools:-SetProperty('K\_p\_min',value,evalf[3](K\_p(omega\_z\_v[minOfUpperBound])));

> DocumentTools:-SetProperty('K\_d\_min',value,evalf[3](K\_d(omega\_z\_v[minOfUpperBound])));

> h\_endless := (t,omega\_z)->evalf(invlaplace(W\_замкнутая(s,omega\_z)/s,s,t)):

> h\_endless\_n := t->evalf(Re(h\_endless(t,omega\_z\_v[minOfUpperBound]))):

> T\_уст\_end := getTransientPeriod2(t->h\_endless(t,omega\_z\_v[minOfUpperBound]),limit(W\_замкнутая(omega\*I,minOfUpperBound),omega=0),h\_endless\_n(0),2000):

> DocumentTools:-SetProperty('t\_ust\_end',caption,T\_уст\_end);

> T\_уст\_end\_3 := evalf[3](3\*T\_уст\_end):

> DocumentTools:-SetProperty('h\_endless\_',value,plot(h\_endless\_n(t),t=0..T\_уст\_end\_3));

>

> f := omega\_z->getTransientPeriod2(t->h\_endless(t,omega\_z),limit(W\_замкнутая(omega\*I,omega\_current),omega=0),h\_endless\_n(0),1000):

> DocumentTools:-SetProperty('t\_ust\_',value,evalf[4](T\_уст));

>

> DocumentTools:-SetProperty('sigma\_',value,evalf[3](getTransientOvershoot(h\_original,W\_исх(0),A)));

> #evalf[3](h\_endless\_n(t));

> DocumentTools:-SetProperty('sigma\_end',caption,getTransientOvershoot(h\_endless\_n,evalf(limit(W\_замкнутая(x,omega\_z\_v[minOfUpperBound]),x=0)),A));

> l:= j -> omega\_z\_beg + j\*((omega\_z\_end - omega\_z\_beg)/numberOfSteps):

omega\_k\_v:=Vector[row](numberOfSteps,l):

> #findMin(map(f,omega\_k\_v));

**Расчет коэффициентов регулятора по критерию минимальной оценки перерегулирования**

> indexOfMinOvershoot := findIndexOfMin(overshoot\_v);

> omegaOfMinOvershoot := omega\_z\_v[indexOfMinOvershoot];

> DocumentTools:-SetProperty('omegaOfMinOvershoot\_',caption,evalf[3](omegaOfMinOvershoot));

> DocumentTools:-SetProperty('R\_s\_omegaOfMinOvershoot',value,evalf[3](R(s,omegaOfMinOvershoot)));

> evalf(omegaOfMinOvershoot);

> evalf[3](rationalize(W\_замкнутая(s,omegaOfMinOvershoot)));

> DocumentTools:-SetProperty('W\_z\_s\_omegaOfMinOvershoot',value,evalf[3](rationalize(W\_замкнутая(s,omegaOfMinOvershoot))));

> DocumentTools:-SetProperty('K\_p\_omegaOfMinOvershoot',value,evalf[3](K\_p(omegaOfMinOvershoot)));

> DocumentTools:-SetProperty('K\_d\_omegaOfMinOvershoot',value,evalf[3](K\_d(omegaOfMinOvershoot)));

> h\_endless\_omegaOfMinOvershoot := t->evalf(Re(h\_endless(t,omegaOfMinOvershoot))):

> T\_уст\_omegaOfMinOvershoot := getTransientPeriod2(t->h\_endless(t,omegaOfMinOvershoot),limit(W\_замкнутая(omega\*I,omegaOfMinOvershoot),omega=0),h\_endless\_omegaOfMinOvershoot(0),2000):

> DocumentTools:-SetProperty('t\_ust\_omegaOfMinOvershoot',caption,T\_уст\_omegaOfMinOvershoot);

> T\_уст\_omegaOfMinOvershoot\_3 := evalf[3](3\*T\_уст\_omegaOfMinOvershoot):

> DocumentTools:-SetProperty('h\_omegaOfMinOvershoot\_',value,plot(h\_endless\_omegaOfMinOvershoot(t),t=0..T\_уст\_omegaOfMinOvershoot\_3));

>

> #f := omega\_z->getTransientPeriod2(t->h\_endless(t,omega\_z),limit(W\_замкнутая(omega\*I,omega\_current),omega=0),h\_endless\_n(0),1000):

> #DocumentTools:-SetProperty('t\_ust\_',value,evalf[4](T\_уст));

>

> #evalf[3](h\_endless\_n(t));

> DocumentTools:-SetProperty('sigma\_omegaOfMinOvershoot',caption,getTransientOvershoot(h\_endless\_omegaOfMinOvershoot,evalf(limit(W\_замкнутая(x,omegaOfMinOvershoot),x=0)),A