Приложение А

(обязательное)

Листинг разработанных скриптов

А1 Решение вариационной задачи для стационарной модели фильтрации

```
> restart:
#-------------------------------------------------ПАРАМЕТРЫ
ЗАДАЧИ-----------------------------------------------------
kNol:=1:#проницаемость в нуле
CantorCoeff:=1:#коэффициент при канторовой лестнице
#краевые условия на 0 и 1
krai0:=1:
kraiL:=2:

#задание канторовой лестницы
n:=6:
N:=2^n-1:
for k from 1 to N do
Ssum:=0:
for i from 1 to 10 do
p:=floor(k/(2^(n-i))):
if irem(p,2)=0 then
b[i]:=0:
else
b[i]:=1:
end if:
Ssum:=Ssum+b[i]/(3^i):
end do:
CantorSet[k]:=2*Ssum:
y[k]:=k/((N+1)):
end do:

#распределение проницаемостей на отрезке решаемой задачи
X[0]:=0:
k[0]:=kNol:
for i from 1 to N do
k[i]:=CantorCoeff*y[i]+kNol:
X[i]:=CantorSet[i]:
end do:
L:=X[N]:

#общее решение при постоянном k
U:=C*x+D:

#задание системы уравнений на определение коэффициентов во всех
уравнениях  на всех отрезках
eq[1]:=d[1]=krai0:
eq[2]:=d[N]=kraiL-c[N]*L:
for i from 1 to N-1 do
```

```
      eq[i+2]:=c[i]*X[i]+d[i]=c[i+1]*X[i]+d[i+1]:
      end do:
      for j from 1 to N-1 do
      eq[j+N-1+2]:=k[j]*c[j]=k[j+1]*c[j+1]:
      end do:

      #решение системы уравнений на определение коэффициентов
      solve({seq(eq[i],i=1..2*N)},{seq(c[i],i=1..N),seq(d[i],i=1..N)}
):
      #запись решения в виде кусочной функции
      UcantorNaotrezke:=seq(subs(subs(%[i],C=c[i]),subs(%[N+i],D=d[i]
),U),i=1..N):
      A  :=  Matrix([[seq(x>=X[i]  and  x<=X[i+1],  i  =  0  ..  N-1)],
[seq(UcantorNaotrezke[i],i=1..N)]]):
      Ucantor:=piecewise(seq(i, i in A)):
      A  :=  Matrix([[seq(x>=X[i]  and  x<=X[i+1],  i  =  0  ..  N-1)],
[seq(diff(UcantorNaotrezke[i],x),i=1..N)]]):
      GradUcantor:=piecewise(seq(i, i in A)):

      #решение краевой задачи для проницаемости=степенной функции
      solverStacionarkaOsred:=proc(a,b,X)
      option remember;
      local Kapprox,r,UApprox,u:
      global CantorCoeff, kNol:
      Kapprox:=CantorCoeff*(x)^a*b+kNol:
      r := proc (aa) evalf(Int(1/(Kapprox),x=0..aa)): end proc:
      #общее решение
      [r(X)*(kraiL-krai0)/r(1)+krai0,(kraiL-
krai0)/(r(1)*subs(x=X,Kapprox))]:
      end proc:

      #построение сетки по пространственной переменной
      nX := 400:
      shagX:=(X[N]-X[0])/nX:
      Xint[0]:=X[0]:
      for k from 1 to nX do
      Xint[k] := Xint[k-1]+shagX:
      end do:
      #построение сетки по степени
      lRightA:=0.9:
      lLeftA:=0.4:
      nA := 30:
      Aint[0]:=lLeftA:
      shagA:=(lRightA-lLeftA)/nA:
      for k from 1 to nA do
      Aint[k] := Aint[k-1]+shagA:
      end do:
      #построение сетки по коэффициенту
      lRightB:=1.5:
      lLeftB:=0.4:
      nB := 10:
      Bint[0]:=lLeftB:
```

```
shagB:=(lRightB-lLeftB)/nB:
for k from 1 to nB do
Bint[k] := Bint[k-1]+shagB:
end do:

#расчет значений функционала
for rr from 1 to nA do
for kk from 1 to nB do
IntSol_x:=0:
for pp from 30 to nX-2 do
IntSol_x:=IntSol_x+((solverStacionarkaOsred(Aint[rr],Bint[kk],X
int[pp])[1]-subs(x=Xint[pp],Ucantor))^2)*(Xint[pp]-Xint[pp-1]):
end do:
```

А2 Решение вариационной задачи для нестационарной модели фильтрации

```
> restart:
#-------------------------------------------------ПАРАМЕТРЫ
ЗАДАЧИ-----------------------------------------------------------
#проницаемость в нуле
kNol:=0:
mNol:=0:
delCantor:=1:

#задание канторовой лестницы
n:=6:
N:=2^n-1:
for k from 1 to N do
Ssum:=0:
for i from 1 to 10 do
p:=floor(k/(2^(n-i))):
if irem(p,2)=0 then
b[i]:=0:
else
b[i]:=1:
end if:
Ssum:=Ssum+b[i]/(3^i):
end do:
CantorSet[k]:=2*Ssum:
y[k]:=k/((N+1)):
end do:
k:='k':

#распределение проницаемостей на отрезке решаемой задачи
X[0]:=0:
k[0]:=kNol:
for i from 1 to N do
k[i]:=y[i]*delCantor+kNol:
X[i]:=CantorSet[i]:
end do:
L:=X[N]:
```

```
     #вид решения
     U:=C*(x^2+2*K*t)+D:

     #Краевая задача Коши для первого уравнения
     Krai0:=t;

     # определение коэффициентов из краевых условий
     CoefficientVector(subs(x=X[0],K=k[1],C=A[1],D=B[1],U)-Krai0,
t):
     CoefU[1]:=solve({%[1]=0, %[2]=0},{A[1],B[1]}):
     # определение коэффициентов из условий сшивки
     for i from 1 to N-1 do
     CoefU[i];
     CoefficientVector(subs(x=X[i],K=k[i],C=A[i],D=B[i],%[1],%[2],U)
-subs(x=X[i],K=k[i+1],C=A[i+1],D=B[i+1],U),t);
     CoefU[i+1]:=solve({%[1]=0, %[2]=0},{A[i+1],B[i+1]}):
     end do:

     #запись решения в виде кусочной функции
     UcantorNaotrezke:=seq(subs(C=A[i],D=B[i],CoefU[i],K=k[i],U),i=1
..N):
     A  :=  Matrix([[seq(x>=X[i]  and  x<=X[i+1],  i  =  0  ..  N-1)],
[seq(UcantorNaotrezke[i],i=1..N)]]):
     Ucantor:=piecewise(seq(i, i in A)):
     A  :=  Matrix([[seq(x>=X[i]  and  x<=X[i+1],  i  =  0  ..  N-1)],
[seq(diff(UcantorNaotrezke[i],x),i=1..N)]]):
     GradUcantor:=piecewise(seq(i, i in A)):
     A  :=  Matrix([[seq(x>=X[i]  and  x<=X[i+1],  i  =  0  ..  N-1)],
[seq(diff(UcantorNaotrezke[i],t),i=1..N)]]):
     DtUcantor:=piecewise(seq(i, i in A)):

     #допостановка смешанной задачи для аппроксимации
     Kosh:=eval(Ucantor,t=0);
     Krai0:=eval(Ucantor,x=X[0]);
     KraiL:=evalf(eval(Ucantor,x=X[N]));

     #решение краевой задачи для проницаемости=степенной функции
     solverStacionarkaOsred:=proc(a,b,c,d)
     option remember;
     global kNol,Krai0,KraiL,Kosh,X,delCantor;
     local Kapprox,Mapprox,PDE,pds,IBC,g,result;
     Kapprox:=(x)^a*b*delCantor+kNol:
     Mapprox:=(x)^c*d+mNol:
     #уравнение, граничные условия, решение встроенным решателем
     PDE  :=  Mapprox*diff(u(x,  t),  t)  =  diff(Kapprox*diff(u(x,  t),x),
x);
     IBC  :=  {u(X[0],  t)  =  Krai0,  u(x,  0)  =  Kosh,  u(X[N],t)  =  KraiL}:
     pds  :=  pdsolve(PDE,  IBC,  numeric,time=t,range=X[0]..X[N]):
     pds:-value(output  =  listprocedure);
     subs(%[3],u(x,  t)):
     end proc:
```

```
#построение сетки по пространству
nX := 400:
shagX:=(X[N]-X[0])/nX:
Xint[0]:=X[0]:
for k from 1 to nX do
Xint[k] := Xint[k-1]+shagX:
end do:
#построение сетки по времени
lRightT:=1:
lLeftT:=0.01:
nT := 10:
Tint[0]:=lLeftT:
shagT:=(lRightT-lLeftT)/nT:
for k from 1 to nT do
Tint[k] := Tint[k-1]+shagT:
end do:
#построение сетки по степени
lRightA:=0.9:
lLeftA:=0.4:
nA := 30:
Aint[0]:=lLeftA:
shagA:=(lRightA-lLeftA)/nA:
for k from 1 to nA do
Aint[k] := Aint[k-1]+shagA:
end do:
#построение сетки по коэффициенту
lRightB:=0.9:
lLeftB:=0.7:
nB := 30:
Bint[0]:=lLeftB:
shagB:=(lRightB-lLeftB)/nB:
for k from 1 to nB do
Bint[k] := Bint[k-1]+shagB:
end do:

#расчет значений функционала
for rr from 0 to nA do
for kk from 0 to nB do
IntSol_xt:=0:
forIntSol:=solverStacionarkaOsred(Aint[rr],Bint[kk],0,0.3):
for hh from 0 to nT-1 do
IntSol_x:=0:
for pp from 0 to nX-1 do
IntSol_x:=IntSol_x+((forIntSol(Xint[pp],Tint[hh])-
eval(Ucantor,[x=Xint[pp],t=Tint[hh]]))^2+
     ((forIntSol(Xint[pp+1],Tint[hh])-
forIntSol(Xint[pp],Tint[hh]))/shagX-
eval(GradUcantor,[x=Xint[pp],t=Tint[hh]]))^2+
     ((forIntSol(Xint[pp],Tint[hh+1])-
forIntSol(Xint[pp],Tint[hh]))/shagT-
eval(DtUcantor,[x=Xint[pp],t=Tint[hh]]))^2)*shagX:
```

```
end do:
IntSol_xt:=IntSol_x+IntSol_x*shagT:
end do:
Funkzional[Aint[rr],Bint[kk]]:=IntSol_xt;
end do:
print(nA-rr):
end do:
```

А3 Построение непрерывно параметризованной кривой Коха и расчет массовой функции по алгоритму Монте-Карло

```
> restart:
with(Statistics):
ChZnPoZap:=3:
ShagX:=evalf[1](1/10^ChZnPoZap);

#углы поворота для вычисления фрактальной функции
Theta[0]:=0: Theta[1]:=Pi/3: Theta[2]:=-Pi/3: Theta[3]:=0:
#матрицы поворота
for i from 0 to 3 do
s[i]:=1/3:                T[i]:=matrix(2,2,[s[i]*cos(Theta[i]),-
s[i]*sin(Theta[i]),s[i]*sin(Theta[i]),s[i]*cos(Theta[i])]):
end do:
#вектор в сторону которого растет фрактал
v0:=matrix(2,1,[1,0]):
#необходимые постоянные для параметризации фрактала
for j from 0 to 3 do
A[j]:=evalf(evalm(add(evalm(T[i]&*v0),i=0..j))[1,1]);
B[j]:=evalf(evalm(add(evalm(T[i]&*v0),i=0..j))[2,1]);
C[j]:=evalf(s[j]*cos(Theta[j])):
E[j]:=evalf(s[j]*sin(Theta[j])):
end do:
A[-1]:=0: B[-1]:=0: C[-1]:=0: E[-1]:=0:

#функция получения массива координат фрактала
KoxaPlolnostu:= proc(t)
local zz,i,eq1,eq2,j,jj,TT:
#составление системы уравнений для вычисления массива координат
фрактала, шагаем по параметру t
j:=1: zz[1]:=0:
while zz[j]<1+t do
eq1[j],eq2[j],TT[j]:=KoxaUrav(zz[j]):
zz[j+1]:=zz[j]+t: j:=j+1:
end do:
subs(x[0.]=0,y[0.]=0,solve({seq(eq1[jj],jj=1..j-
1),seq(eq2[jj],jj=1..j-1)},{seq(x[zz[jj]],jj=1..j-
1),seq(y[zz[jj]],jj=1..j-1)}));
#возвращаем координату x,y и парметр t для них
subs(%,[seq([x[zz[jj]],y[zz[jj]],zz[jj]],jj=1..j-1)]);
end proc:
```

```
     #функция получения уравнения на координаты в зависимости от
параметра t
     KoxaUrav := proc(t)
     option remember;
     local eq1,eq2,i,T,qq,k:
     T:=t: k:=4:
     qq:=k*T-trunc(k*T):
     eq1:=x[t]=A[trunc(k*T)-1]+C[trunc(k*T)]*x[qq]-
E[trunc(k*T)]*y[qq]:
     eq2:=y[t]=B[trunc(k*T)-
1]+E[trunc(k*T)]*x[qq]+C[trunc(k*T)]*y[qq]:
     eq1,eq2,T;
     end proc:

     SaveFractalNotWar:=KoxaPlolnostu(ShagX):
     ChisloUzlovFractala:=nops(SaveFractalNotWar)-1:
     KoxaFunction:= proc(t)
     option remember;
     local i:
     for i from 0 to ChisloUzlovFractala do
     if t=i/ChisloUzlovFractala then
     RETURN([SaveFractalNotWar[i+1,1],SaveFractalNotWar[i+1,2]]);
     fi:
     od:
     NULL:
     end proc:

     #параметры начального подразбиения
     delta:=evalf(0.1): SubdivizionTrue[1]:=0: i:=1:
     #начальное подразбиение
     while SubdivizionTrue[i]<1 do
     SubdivizionTrue[i+1]:=evalf[2](SubdivizionTrue[i]+delta/4):
     i:=i+1:
     od:
     N:=i:

     k:=0:
     while k<300 do
     i:='i': j:='j':
     SubdivizionShtrih1:=0:
     #модификация подразбиения, генерация случайного промежутка
     x1:=evalf(Sample(RandomVariable(Uniform(0, 1)), 1)[1]):
     y1:=evalf(Sample(RandomVariable(Uniform(0, 1)), 1)[1]):

     #случайный промежуток не должен быть точкой
     while evalf[5](x1)=evalf[5](y1) do
     y1:=evalf(Sample(RandomVariable(Uniform(0, 1)), 1)[1]):
     od:
     y:=max(x1,y1); x:=min(x1,y1);
     #точки подразбиения из случайного промежутка
     j:=1:
     for i from 1 to N do
```

```
      if SubdivizionTrue[i]>=x and SubdivizionTrue[i]<=y then
      SubdivizionShtrih1[j]:=SubdivizionTrue[i]:
      j:=j+1:
      #запомним номер с которого будем изменять разбиение
      if j=2 then
      iStart:=i:
      fi:
      fi:
      od:
      #сколько узлов, которые нужно изменить
      m:=j-3:
      #запомним номер узла до которого будем изменять разбиение
      iFinish:=iStart+m+1:

      keyOnlyOneWay:=0:
      #случайный промежуток должен содержать хотя бы 3 точки из
начального подразбиения
      if m>1 then
      #модификация A
      RandomVariableForProbability:=Sample(RandomVariable(Uniform(0,
1)), 1)[1]:
      pC:=min(1,delta/(y-x));
      if  RandomVariableForProbability<evalf(pC)  and  keyOnlyOneWay=0
then
      #сдвигаем все элементы (кроме граничных) подразбиения случайным
образом
      key:=-1:
      while key<>SubdivizionShtrih1[m+2] do
      SubdivizionShtrih1Shift:=0:
      SubdivizionShtrih1Shift[1]:=SubdivizionShtrih1[1]:
      RandomVariableForShift:=Sample(RandomVariable(Uniform(-
delta/10, delta/10)), m+5):
      for i from 2 to m+1 do
      SubdivizionShtrih1Shift[i]:=abs(evalf[5](trunc(10^ChZnPoZap*(Su
bdivizionShtrih1[i]+RandomVariableForShift[i]))/10^ChZnPoZap)):
      od:
      SubdivizionShtrih1Shift[m+2]:=SubdivizionShtrih1[m+2]:
      SubdivizionShtrih1Shift:=sort([seq(SubdivizionShtrih1Shift[pp],
pp=1..m+2)]);
      key:=SubdivizionShtrih1Shift[m+2]:
      od:

      #проверка диаметра модификации разбиения
diamShtrih1Shift:=evalf(max(seq(SubdivizionShtrih1Shift[ii]-
SubdivizionShtrih1Shift[ii-1],ii=2..m+2))):
      if diamShtrih1Shift<=delta then
      for i from 1 to m+2 do
      SubdivizionShtrih2[i]:=SubdivizionShtrih1Shift[i]:
      od:

      #запись нового подразбиения на всем отрезке
      for i from 1 to iStart do
```

```
      Subdivizion1[i]:=SubdivizionTrue[i]:
      od:
      j:=2:
      for i from iStart+1 to iFinish-1 do
      Subdivizion1[i]:=SubdivizionShtrih2[j]:
      j:=j+1:
      od:

      for i from iFinish to N do
      Subdivizion1[i]:=SubdivizionTrue[i]:
      od:

      #расчет массовой функции и проверка ее уменьшения
      sigma1:=add(evalf((sqrt((KoxaFunction(Subdivizion1[jj+1])[1]-
KoxaFunction(Subdivizion1[jj])[1])^2+(KoxaFunction(Subdivizion1[jj+1
])[2]-KoxaFunction(Subdivizion1[jj])[2])^2))^(ln(4)/ln(3))),jj=1..N-
1):
      sigma:=add(evalf((sqrt((KoxaFunction(SubdivizionTrue[jj+1])[1]-
KoxaFunction(SubdivizionTrue[jj])[1])^2+(KoxaFunction(SubdivizionTru
e[jj+1])[2]-
KoxaFunction(SubdivizionTrue[jj])[2])^2))^(ln(4)/ln(3))),jj=1..N-1):
      if sigma1<sigma then
      k:=k+1:
      for i from 1 to N do
      SubdivizionTrue[i]:=Subdivizion1[i]:
      od:
      keyOnlyOneWay:=1:
      fi:
      fi:
      fi:

      #вариант В
      RandomVariableForProbability:=Sample(RandomVariable(Uniform(0,
1)), 1)[1]:
      pD:=min(1,delta/(y-x));
      if  RandomVariableForProbability<evalf(pD)  and  keyOnlyOneWay=0
then
      #удаляем все элементы (кроме граничных) подразбиения
      SubdivizionShtrih1Delete[1]:=SubdivizionShtrih1[1]:
      SubdivizionShtrih1Delete[2]:=SubdivizionShtrih1[m+2]:
      #проверка диаметра модификации разбиения
      diamShtrih1Delete:=evalf(SubdivizionShtrih1Delete[2]-
SubdivizionShtrih1Delete[1]):
      if diamShtrih1Delete<=delta then
      SubdivizionShtrih2[1]:=SubdivizionShtrih1Delete[1]:
      SubdivizionShtrih2[2]:=SubdivizionShtrih1Delete[2]:
      #запись нового подразбиения на всем отрезке
      for i from 1 to iStart-1 do
      Subdivizion1[i]:=SubdivizionTrue[i]:
      od:
      Subdivizion1[iStart]:=SubdivizionShtrih2[1]:
      Subdivizion1[iStart+1]:=SubdivizionShtrih2[2]:
```

```
      jjj:=iStart+2:
      for i from iFinish+1 to N do
      Subdivizion1[jjj]:=SubdivizionTrue[i]:
      jjj:=jjj+1:
      od:
      jjj:=jjj-1:

      #расчет массовой функции и проверка ее уменьшения
      sigma1:=add(evalf((sqrt((KoxaFunction(Subdivizion1[jj+1])[1]-
KoxaFunction(Subdivizion1[jj])[1])^2+(KoxaFunction(Subdivizion1[jj+1
])[2]-
KoxaFunction(Subdivizion1[jj])[2])^2))^(ln(4)/ln(3))),jj=1..jjj-1):
      sigma:=add(evalf((sqrt((KoxaFunction(SubdivizionTrue[jj+1])[1]-
KoxaFunction(SubdivizionTrue[jj])[1])^2+(KoxaFunction(SubdivizionTru
e[jj+1])[2]-
KoxaFunction(SubdivizionTrue[jj])[2])^2))^(ln(4)/ln(3))),jj=1..N-1):
      if sigma1<sigma then
      k:=k+1:
      SubdivizionTrue:=0:
      for i from 1 to jjj do
      SubdivizionTrue[i]:=Subdivizion1[i]:
      od:#1 to j
      N:=jjj:
      keyOnlyOneWay:=0:
      fi:
      fi:
      keyOnlyOneWay:=1:
      fi:

      #вариант С
      RandomVariableForProbability:=Sample(RandomVariable(Uniform(0,
1)), 1)[1]:
      pI:=min(1,delta/(y-x));
      if  RandomVariableForProbability<evalf(pI)  and  keyOnlyOneWay=0
then
      #дополняем подразбиение
      i:=1: j:=1:
      while j<=m do
      if SubdivizionShtrih1[j+1]-SubdivizionShtrih1[j]>delta/10 then
      SubdivizionShtrih1Insert[i]:=SubdivizionShtrih1[j]: i:=i+1:
      SubdivizionShtrih1Insert[i]:=SubdivizionShtrih1Insert[i-1]:
      while                              abs(SubdivizionShtrih1Insert[i]-
SubdivizionShtrih1Insert[i-1])=0 do
      SubdivizionShtrih1Insert[i]:=evalf[5](trunc(10^ChZnPoZap*(Sampl
e(RandomVariable(Uniform(SubdivizionShtrih1[j],
SubdivizionShtrih1[j+1])), 1)[1]))/10^ChZnPoZap):
      od:
      i:=i+1:
      else
      SubdivizionShtrih1Insert[i]:=SubdivizionShtrih1[j]:
      i:=i+1:
      fi:
```

```
      j:=j+1:
      od:
      SubdivizionShtrih1Insert[i]:=SubdivizionShtrih1[j]:
      NN:=i;

      #проверка диаметра модификации разбиения
      diamShtrih1Insert:=evalf(max(seq(SubdivizionShtrih1Insert[ii]-
SubdivizionShtrih1Insert[ii-1],ii=2..NN))):
      if diamShtrih1Insert<=delta then
      for i from 1 to NN do
      SubdivizionShtrih2[i]:=SubdivizionShtrih1Insert[i]:
      end do:

      #запись нового подразбиения на всем отрезке
      for i from 1 to iStart do
      Subdivizion1[i]:=SubdivizionTrue[i]:
      od:
      j:=iStart:
      for i from 1 to NN do
      Subdivizion1[j]:=SubdivizionShtrih2[i]: j:=j+1:
      end do:
      for i from iFinish to N do
      Subdivizion1[j]:=SubdivizionTrue[i]: j:=j+1:
      od:
      j:=j-1:

      #расчет массовой функции и проверка ее уменьшения
      sigma1:=add(evalf((sqrt((KoxaFunction(Subdivizion1[jj+1])[1]-
KoxaFunction(Subdivizion1[jj])[1])^2+(KoxaFunction(Subdivizion1[jj+1
])[2]-KoxaFunction(Subdivizion1[jj])[2])^2))^(ln(4)/ln(3))),jj=1..j-
1):
      sigma:=add(evalf((sqrt((KoxaFunction(SubdivizionTrue[jj+1])[1]-
KoxaFunction(SubdivizionTrue[jj])[1])^2+(KoxaFunction(SubdivizionTru
e[jj+1])[2]-
KoxaFunction(SubdivizionTrue[jj])[2])^2))^(ln(4)/ln(3))),jj=1..N-1):
      if sigma1<sigma then
      k:=k+1: SubdivizionTrue:=0:
      for i from 1 to j do
      SubdivizionTrue[i]:=Subdivizion1[i]:
      od:
      N:=j:
      fi:
      keyOnlyOneWay:=1:
      fi:
      fi:
      fi:
      keyOnlyOneWay:=0:
      od:
```