Конечнозонные решения КдФ

К лекции 6 (2024)

---

1 Кноидальная волна

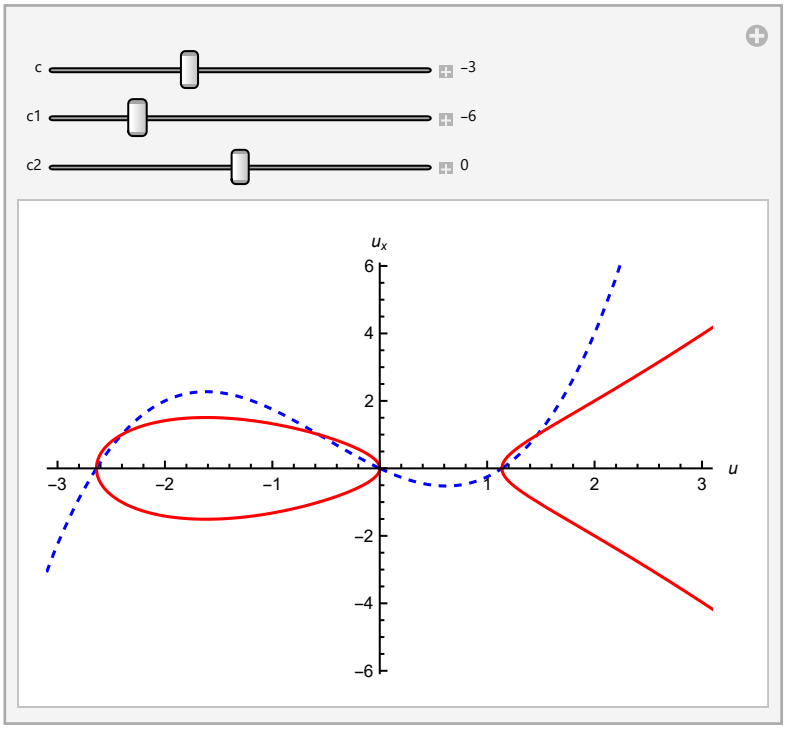
Решения КдФ в виде бегущей волны приводят к уравнению

(1) $u_x^2 = P(u) = 2\,u^3 - c\,u^2 + c_1\,u + c_2.$

Оно определяет кривую в фазовой плоскости (u, u_x). Форма кривой определяется нулями правой части. Если они вещественны и различны, то имеем овал и неограниченную ветвь. Если один корень вещественный и два комплексно сопряженных, то есть только одна неограниченная ветвь. На следующем графике можно посмотреть, как меняется кривая в зависимости от выбора коэффициентов (пунктиром нарисован график самого кубического многочлена P).

In[1]:=
```
Clear[u, k, b, c, c1, c2, d]

plot[P_] := Plot[{P, Sqrt[P], -Sqrt[P]}, {u, -4, 4},
  PlotStyle → {{Dashed, Blue}, Red, Red},
  PlotRange → {{-3.1, 3.1}, {-6.1, 6.1}},
  AxesLabel → {"u", "u_x"}]

Manipulate[
 plot[(2 u^3 - c u^2 + c1 u + c2)/4],
 {{c, -3}, -10, 10, Appearance → "Labeled"},
 {{c1, -6}, -10, 10, Appearance → "Labeled"},
 {{c2, 0}, -10, 10, Appearance → "Labeled"}
]
```

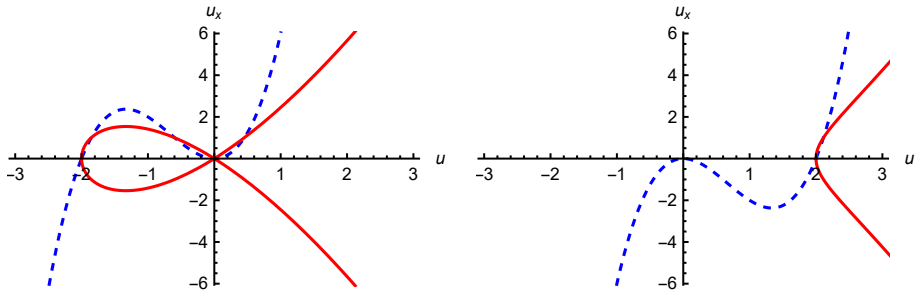
Out[3]=



Движению по неограниченным ветвям отвечают, в плоскости (x, u), полюсные решения, движению по овалам – ограниченные решения. Варьируя параметры, можно получить и вырожденные случаи с кратными корнями. Движению по петле на левом рисунке отвечает солитон.

In[4]:=
```
GraphicsRow[{plot[2 u^2 (u + 2)], plot[2 u^2 (u - 2)]}, ImageSize → 500]
```

Out[4]=



Рассмотрим движение по ограниченному овалу кривой в случае, когда все корни вещественны и различны. Уравнение для u совпадает, с точностью до линейной замены, с уравнением для $\wp$-функции Вейерштрасса, но, так как эта функция имеет полюс в нуле, то удобнее выразить ответ через эллиптические функции Якоби. Напомним их определение: это решение системы ОДУ

(2) $\mathrm{sn}'\,x = \mathrm{dn}\,x\,\mathrm{cn}\,x,$
$\mathrm{cn}'\,x = -\mathrm{dn}\,x\,\mathrm{sn}\,x,$
$\mathrm{dn}'\,x = -k\,\mathrm{sn}\,x\,\mathrm{cn}\,x,$
$\mathrm{sn}\,0 = 0,\ \ \mathrm{cn}\,0 = \mathrm{dn}0 = 1,$

где $0 \le k \le 1$. Из этих уравнений легко получить соотношения

$$\mathrm{sn}^2\,x + \mathrm{cn}^2\,x = 1,\ \ k\,\mathrm{sn}^2\,x + \mathrm{dn}^2\,x = 1.$$

Эти функции уже встроены в *Mathematica*. При $k = 0$ получаем обычную тригонометрию. При $k = 1$ приходим к солитону.

In[5]:=
```
{cn, sn, dn} = {JacobiCN[x, k], JacobiSN[x, k], JacobiDN[x, k]}
% /. k → 0
%% /. k → 1
```

Out[5]= {JacobiCN[x, k], JacobiSN[x, k], JacobiDN[x, k]}

Out[6]= {Cos[x], Sin[x], 1}

Out[7]= {Sech[x], Tanh[x], Sech[x]}

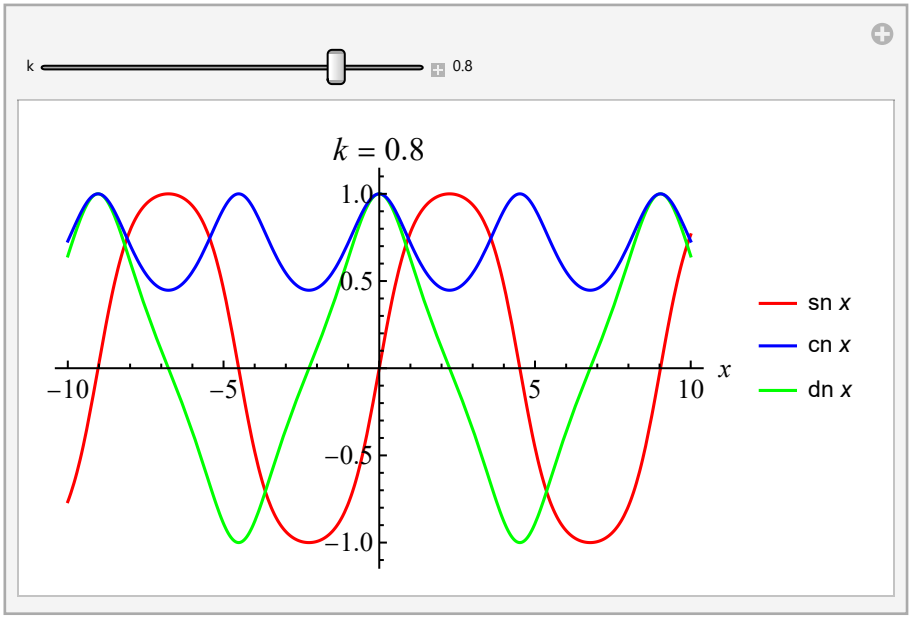
Проверка системы и первых интегралов

In[8]:=
```
{D[sn, x] - dn cn,
 D[cn, x] + dn sn ,
 D[dn, x] + k sn cn,
 D[sn^2 + cn^2, x],
 D[k sn^2 + dn^2, x]}
```

Out[8]= {0, 0, 0, 0, 0}

In[9]:=
```
plot1[kk_] := Plot[Evaluate[{sn, cn, dn} /. k → kk], {x, -10, 10},
  PlotStyle → {Red, Green, Blue, Black, Black, Black},
  PlotRange → {-1.15, 1.15},
  PlotLegends → LineLegend[{Red, Blue, Green}, {"sn x", "cn x", "dn x"}],
  AxesLabel → {"x ", None},
  PlotLabel → "k = " <> ToString[kk],
  BaseStyle → {FontSize → 14, FontFamily → "Times New Roman"}]

Manipulate[plot1[k],
 {{k, 0.8}, 0, 1, Appearance → "Labeled"}]
```

Out[10]=



Преобразовние от системы (2) к уравнению (1): нетрудно показать, что если положить $u = K_1 + K_2\,\mathrm{cn}^2(b\,x)$, то u_x^2 будет представляться в виде многочлена третей степени от u. Остаётся только немного уточнить константы: K_1, K_2, c_1, c_2 выразим через b, c, k.

In[11]:=
```
sub = {JacobiDN[X_, k]^2 :> 1 - k JacobiSN[X, k]^2, JacobiCN[X_, k]^2 :> 1 - JacobiSN[X, k]^2};

U = K1 + K2 JacobiCN[b (x - c t) + d, k]^2;
Collect[D[U, x]^2 - 2 U^3 + c U^2 - c1 U - c2 /. sub, JacobiSN[_, _], Factor]

CoefficientList[%, JacobiSN[d + b (-c t + x), k]] / K2^2;

Solve[% == 0, {K1, K2, c1, c2}]
```

Out[13]=
$$-c2 - c1\,K1 + c\,K1^2 - 2\,K1^3 - c1\,K2 + 2\,c\,K1\,K2 - 6\,K1^2\,K2 + c\,K2^2 - 6\,K1\,K2^2 - 2\,K2^3 + K2\left(c1 - 2\,c\,K1 + 6\,K1^2 + 4\,b^2\,K2 - 2\,c\,K2 + 12\,K1\,K2 + 6\,K2^2\right)\text{JacobiSN}[d + b\,(-c\,t + x), k]^2 - K2^2\left(4\,b^2 - c + 4\,b^2\,k + 6\,K1 + 6\,K2\right)\text{JacobiSN}[d + b\,(-c\,t + x), k]^4 + 2\,K2^2\left(2\,b^2\,k + K2\right)\text{JacobiSN}[d + b\,(-c\,t + x), k]^6$$

Out[15]=
$$\left\{\left\{K1 \to \frac{1}{6}\left(-4\,b^2 + c + 8\,b^2\,k\right),\ K2 \to -2\,b^2\,k,\ c1 \to \frac{1}{6}\left(-16\,b^4 + c^2 + 16\,b^4\,k - 16\,b^4\,k^2\right),\ c2 \to \frac{1}{108}\left(-128\,b^6 + 48\,b^4\,c - c^3 + 192\,b^6\,k - 48\,b^4\,c\,k + 192\,b^6\,k^2 + 48\,b^4\,c\,k^2 - 128\,b^6\,k^3\right)\right\}\right\}$$

Итак, получили такое решение, сделаем проверку, что оно действительно удовлетворяет КдФ.

In[16]:=
```
U = (4 b^2 (2 k - 1) + c)/6 - 2 b^2 k JacobiCN[b (x - c t) + d, k]^2;
Factor[Factor[D[u, t] - D[u, x, x, x] + 6 u D[u, x]] /. sub]
```

Out[17]=
```
0
```

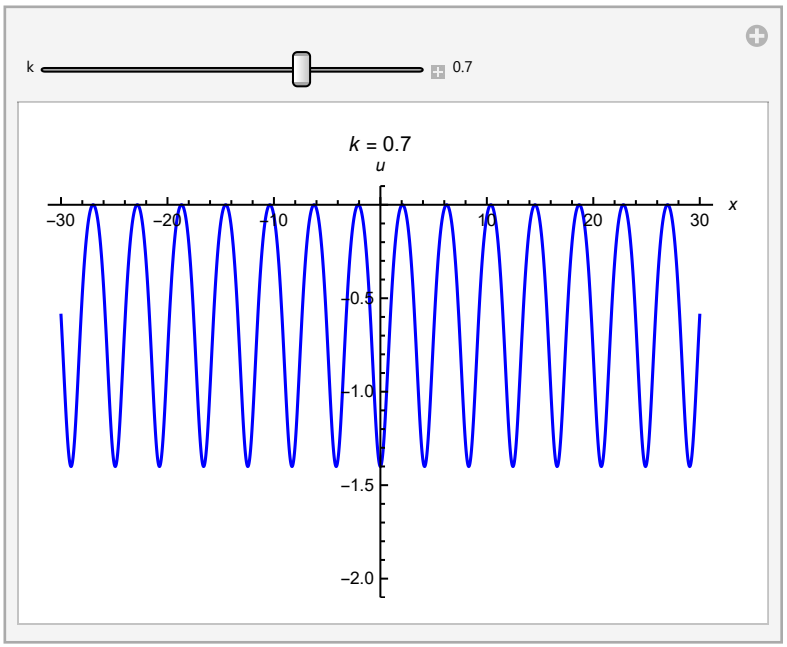
Постоянную c можно выбрать так, чтобы уничтожить свободный член, благодаря преобразованию Галилея, тогда график будет касаться оси x. Далее, за счет растяжения, можно положить $b = 1$; параметр d уничтожается сдвигом x или t. Остаётся один существенный параметр k. При его увеличении пички в кноидальной волне разъезжаются друг от друга и в пределе $k = 1$ получается солитонное решение.

In[18]:=
```
U = -2 k JacobiCN[x + 8 k t, k]^2;
Factor[Factor[D[u, t] - D[u, x, x, x] + 6 u D[u, x]] /. sub]
```

Out[19]=
```
0
```

In[20]:=
```
plot2[k_] := Plot[-2 k JacobiCN[x, k]^2, {x, -30, 30}, PlotStyle → Blue,
  PlotRange → {0.1, -2.1}, AxesLabel → {"x", "u"}, PlotLabel → "k = " <> ToString[k]]
Manipulate[plot2[k],
 {{k, 0.7}, 0, 1, Appearance → "Labeled"}]
```

Out[21]=



2 Двухзонное решение

Численное построение двухзонного, или двухфазного решения КдФ. Для этого решаются совместные уравнения (1) и (2), определяющие эволюцию по x и t. Уравнение (1) – это один раз проинтегрированное стационарное уравнение для высшей симметрии КдФ пятого порядка плюс младшие симметрии с константами c_i. На этом примере мы проверяем, непосредственно в исходных динамических переменных u_n, что такие стационарные уравнения приводят к паре коммутирующих векторных полей. Это и позволяет использовать их для построения решений. Более эффективный численный счет получается при переходе к уравнениям Дубровина, которые дают также аналитический метод решения, но в этой программе мы этого не касаемся, см. текст лекции.

Задача. Пусть $u_n = \partial_x^n(u)$. Проверить, символически и численно, что уравнение Новикова

(1) $u_4 - 10\,u\,u_2 - 5\,u_1^2 + 10\,u^3 + c_1\left(u_2 - 3\,u^2\right) + c_2\,u + c_3 = 0,$

где c_1, c_2, c_3 произвольные постоянные, совместно с уравнением КдФ

(2) $u_t = u_3 - 6\,u\,u_1.$

Если это действительно так, построить график какого-нибудь решения $u(x,\,t)$ удовлетворяющего обоим уравнениям.

Решение. Первый способ символической проверки. Определим эволюционное дифференцирование D_t на бесконечном наборе u_n и проверим выполнение тождества

$$D_t(G) = \left(D_x^3 - 6\,u\,D_x\right)(G),$$

где G – левая часть (1). Тогда, если $G = 0$, то и $D_t(G) = 0$, что и требуется.

In[22]:=
```
dif[f_] := Block[{a}, D[f /. u[n_] :> u[n] + a du[n], a] /. a → 0]
dx[f_] := dif[f] /. du[n_] :> u[n + 1]
dx[f_, n_] := Nest[dx, f, n]

ut = u[3] - 6 u[0] × u[1];
dt[f_] := dif[f] /. du[n_] :> dx[ut, n]

G = u[4] - 10 u[0] × u[2] - 5 u[1]^2 + 10 u[0]^3 + c1 (u[2] - 3 u[0]^2) + c2 u[0] + c3;
Expand[dt[G] - dx[G, 3] + 6 u[0] × dx[G]]
```

Out[28]=
```
0
```

Второй способ. Перепишем оба уравнения в виде конечномерных динамических систем. Так как четвертая производная выражается из ОДУ (1) через младшие, то динамическими переменными являются $u = u_0$, u_1, u_2, u_3. Сразу можно написать векторное поле D_x (точнее – то, как оно действует на произвольную функцию от динамических переменных):

In[29]:=
```
dx4[f_] := u[1] × D[f, u[0]] + u[2] × D[f, u[1]] + u[3] × D[f, u[2]] +
   (10 u[0] × u[2] + 5 u[1]^2 - 10 u[0]^3 - c1 (u[2] - 3 u[0]^2) - c2 u[0] - c3) D[f, u[3]]
dx4[f_, n_] := Nest[dx4, f, n]
```

С (1) справились. С КдФ немного посложнее. Это уравнение задаёт правило дифференцирования по t только для переменной u_0, а нам нужно ещё для u_1, u_2, u_3. Значит, нужно *продолжить* дифференцирование D_t на эти производные, пользуясь тем, что D_x уже определено и полагая, по определению, что $D_t(u_k) = D_x^k(u_t)$. Так мы и делали раньше при определении D_t, сейчас лишь заменяем D_x на укороченное векторное поле, действующее на четырех переменных. Имеем:

In[31]:=
```
dt[f_] := ut D[f, u[0]] + dx4[ut] × D[f, u[1]] + dx4[ut, 2] × D[f, u[2]] + dx4[ut, 3] × D[f, u[3]]
```

Можно посмотреть, что за векторное поле получилось. Вот его компоненты:

In[32]:=
```
dt[u[0]]
Collect[dt4[u[1]], {u[3], u[2], u[1], u[0]}, Factor]
Collect[dt4[u[2]], {u[3], u[2], u[1], u[0]}, Factor]
Collect[dt4[u[3]], {u[3], u[2], u[1], u[0]}, Factor]
```

Out[32]=
```
-6 u[0] × u[1] + u[3]
```

Out[33]=
```
-c3 - c2 u[0] + 3 c1 u[0]^2 - 10 u[0]^3 - u[1]^2 + (-c1 + 4 u[0]) u[2]
```

Out[34]=
```
(-c2 + 6 c1 u[0] - 30 u[0]^2) u[1] + 2 u[1] × u[2] + (-c1 + 4 u[0]) u[3]
```

Out[35]=
```
c1 c3 + (c1 c2 - 4 c3) u[0] + (-3 c1^2 - 4 c2) u[0]^2 + 22 c1 u[0]^3 - 40 u[0]^4 +
  (c1 - 40 u[0]) u[1]^2 + (c1^2 - c2 - 8 c1 u[0] + 10 u[0]^2) u[2] + 2 u[2]^2 + 6 u[1] × u[3]
```

Итак, построили два векторных поля. Совместность означает, что они коммутируют. Считаем коммутатор. Для этого нужно вычислить его значения на координатных функциях (= динамических переменных), то есть, $[D_x, D_t](u_k)$, $k = 0, 1, 2, 3$.

In[36]:=
```
Table[Expand[dx4[dt4[u[k]]] - dt4[dx4[u[k]]]], {k, 0, 3}]
```

Out[36]=
```
{0, 0, 0, 0}
```

Всё сошлось. На самом деле, ясно, что равенство нулю коммутатора на первых трёх координатах выполняется автоматически – в силу построения дифференцирования D_t по формуле $D_t(u_k) = D_x^k(u_t)$. Для примера, испортим что-то в уравнениях. Тогда коммутативность нарушится, но только на последней координате. Так что, можно было бы проверять только её, но лишний контроль не мешает.

In[37]:=
```
(* портим ut *)
ut = u[3] - 66 u[0] × u[1];
Table[Expand[dx4[dt4[u[k]]] - dt4[dx4[u[k]]]], {k, 0, 3}]

(* откатываем назад *)
ut = u[3] - 6 u[0] × u[1];
Table[Expand[dx4[dt4[u[k]]] - dt4[dx4[u[k]]]], {k, 0, 3}]
```

Out[38]=
```
{0, 0, 0, 300 c3 u[1] + 300 c2 u[0] × u[1] - 900 c1 u[0]^2 u[1] + 3000 u[0]^3 u[1] -
  900 u[1]^3 + 120 c1 u[1] × u[2] - 1200 u[0] × u[1] × u[2] - 600 u[2] × u[3]}
```

Out[40]=
```
{0, 0, 0, 0}
```

Переходим к численному счёту. Определяем решения задач Коши по x и по t. Для этого к обозначениям функций нужно добавить один аргумент.

In[41]:=
```
nsolx[{u00_, u10_, u20_, u30_}, {C1_, C2_, C3_}, {x0_, x1_}] :=
 NDSolve[
   {u[0]'[x] == u[1][x],
    u[1]'[x] == u[2][x],
    u[2]'[x] == u[3][x],
    u[3]'[x] == (dx4[u[3]] /. u[k_] → u[k][x] /. {c1 → C1, c2 → C2, c3 → C3}),
    u[0][x0] == u00,
    u[1][x0] == u10,
    u[2][x0] == u20,
    u[3][x0] == u30},
   {u[0], u[1], u[2], u[3]},
   {x, x0, x1}][[1]]
```

In[42]:=
```
nsolt[{u00_, u10_, u20_, u30_}, {C1_, C2_, C3_}, {t0_, t1_}] :=
 NDSolve[
   Join[
    Thread[{u[0]'[t], u[1]'[t], u[2]'[t], u[3]'[t]} ==
      Expand[dt4[{u[0], u[1], u[2], u[3]}] /. u[k_] → u[k][t] /. {c1 → C1, c2 → C2, c3 → C3}]],
    {u[0][t0] == u00,
     u[1][t0] == u10,
     u[2][t0] == u20,
     u[3][t0] == u30}],
   {u[0], u[1], u[2], u[3]},
   {t, t0, t1}][[1]]
```

Решаем задачу Коши для (5) от $x = x_0$ до x_1. Значения вектора (u_0, u_1, u_2, u_3) в промежуточных точках, с некоторым шагом δ, принимаем в качестве начальных условий для (6). Решаем соответствующие

задачи Коши от $t = t_0$ до t_1. Потом делаем то же самое в другом порядке, с шагом ϵ по t. Сравниваем получившиеся сетки. Если значения совпадают – значит функция $u(x, t)$ определена корректно, строим её 3D график по полученной сетке.

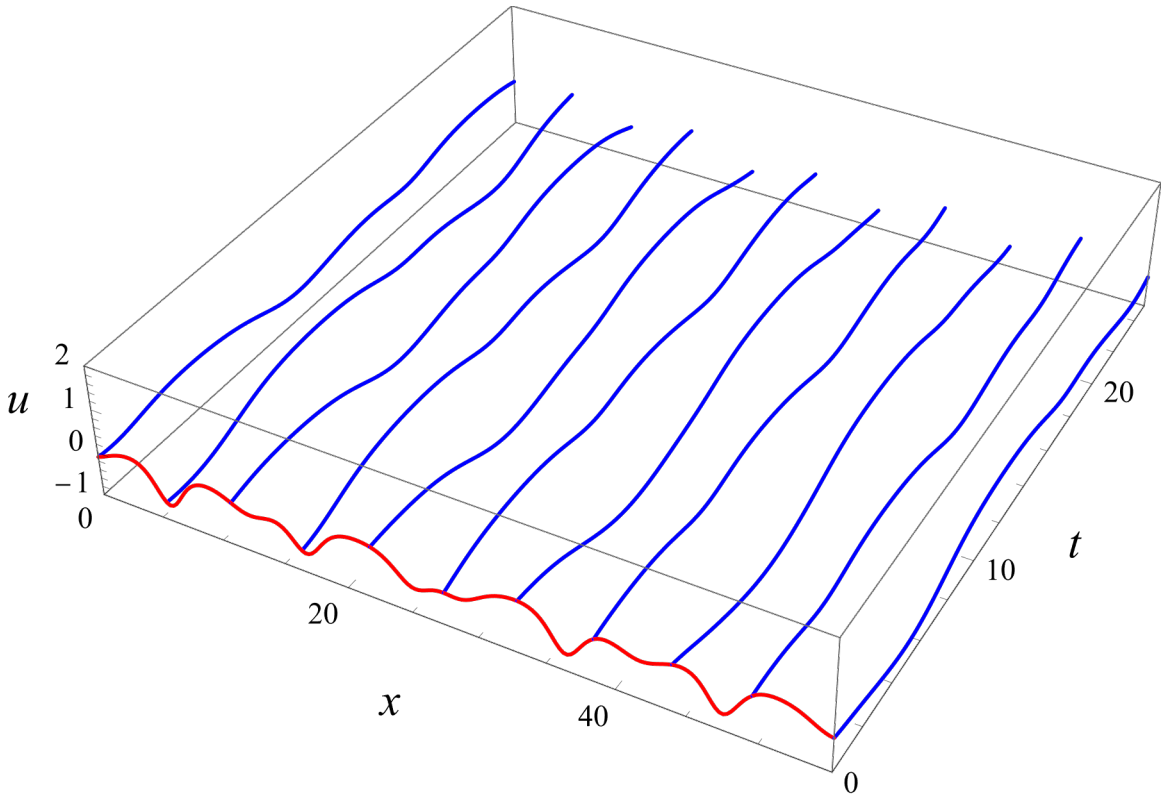
Следует отметить, что в этой задаче многие решения имеют полюсные особенности. Ниже, значения параметров и начальных условий подобраны так, чтобы их не было.

In[43]:=
```
x0 = 0; x1 = 55; (* интервал по x *)
Mx = 200; (* число точек в сетке по x для 3D графика *)
mx = 10; (* число точек для линейчатых графиков *)

t0 = 0; t1 = 25; (* интервал по t *)
Mt = 200; (* число точек в сетке по t для 3D графика *)
mt = 10; (* число точек для линейчатых графиков *)

cc = {0.15, -0.7, 0}; (* параметры *)
U0 = {-0.1, 0.1, 0.1, -0.125}; (* начальные данные при x0, t0 *)
```

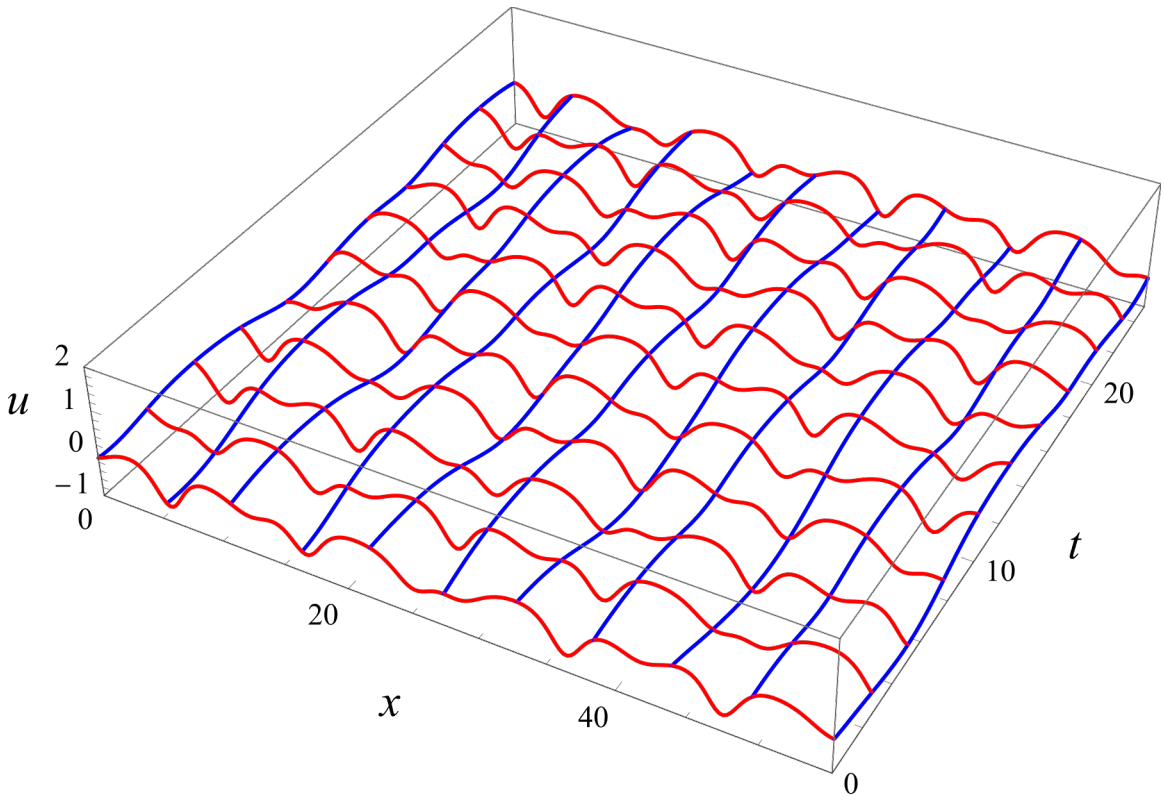
Сначала решим на редкой сетке и построим графики в виде набора кривых. Этого делается просто для наглядности, чтобы получить картинку как на лекции.

In[51]:=
```
(* решаем систему по t;
решаем систему по x для последовательности значений t[0]= t0, ..., t[mt]=t1;
результат сохраняем в виде графического примитива Line *)

nst = nsolt[U0, cc, {t0, t1}];

nstx1 = Table[
    ns = nsolx[
      {u[0][t], u[1][t], u[2][t], u[3][t]} /. t → t0 + j (t1 - t0) / mt /. nst, cc, {x0, x1}];
    Line[Table[
      {x0 + i (x1 - x0) / Mx, t0 + j (t1 - t0) / mt, u[0][x0 + i (x1 - x0) / Mx] /. ns}, {i, 0, Mx}]],
    {j, 0, mt}];

(* делаем то же самое, меняя ролями x и t *)
nsx = nsolx[U0, cc, {x0, x1}];

nsxt1 = Table[
    ns = nsolt[
      {u[0][x], u[1][x], u[2][x], u[3][x]} /. x → x0 + i (x1 - x0) / mx /. nsx, cc, {t0, t1}];
    Line[Table[
      {x0 + i (x1 - x0) / mx, t0 + j (t1 - t0) / Mt, u[0][t0 + j (t1 - t0) / Mt] /. ns}, {j, 0, Mt}]],
    {i, 0, mx}];

gr0 = Graphics3D[{Thick, Red, nstx1[[1]], Blue, nsxt1},
   PlotRange → {{x0, x1}, {t0, t1}, {-1, 2}},
   Axes → True,
   AxesLabel → Evaluate[Style[#, Italic, 24] & /@ {"x", "t", "u"}],
   ImageSize → 600,
   BoxRatios → {1, 1, 0.2},
   BaseStyle → {FontFamily → "Times", FontSize → 16}]

gr1 = Graphics3D[{Thick, Red, nstx1, Blue, nsxt1},
   PlotRange → {{x0, x1}, {t0, t1}, {-1, 2}},
   Axes → True,
   AxesLabel → Evaluate[Style[#, Italic, 24] & /@ {"x", "t", "u"}],
   ImageSize → 600,
   BoxRatios → {1, 1, 0.2},
   BaseStyle → {FontFamily → "Times", FontSize → 16}]
```

Out[55]=



Out[56]=



Красные кривые (эволюция по x) пересекают синие (эволюция по t), что визуально подтверждает коммутативность потоков.

Теперь повторим это же вычисление с более мелким шагом и построим график в виде поверхности.

In[57]:=
```
(* решаем систему по t;
решаем систему по x для последовательности значений t[0]= t0, ..., t[Mt]=t1;
сохраняем результат в виде 2D массива *)

nst = nsolt[U0, cc, {t0, t1}];
nstx2 = Table[
   ns = nsolx[
     {u[0][t], u[1][t], u[2][t], u[3][t]} /. t → t0 + j (t1 - t0) / Mt /. nst, cc, {x0, x1}];
   Table[u[0][x0 + i (x1 - x0) / Mx] /. ns, {i, 0, Mx}], {j, 0, Mt}];

(* делаем то же самое, меняя ролями x и t *)
nsx = nsolx[U0, cc, {x0, x1}];
nsxt2 = Table[
   ns = nsolt[
     {u[0][x], u[1][x], u[2][x], u[3][x]} /. x → x0 + i (x1 - x0) / Mx /. nsx, cc, {t0, t1}];
   Table[u[0][t0 + j (t1 - t0) / Mt] /. ns, {j, 0, Mt}], {i, 0, Mx}];
```

Сравниваем две полученные сетки. Значения, вычисленные двумя способами, совпадают с точностью до численных ошибок, что еще раз подтверждает коммутативность.

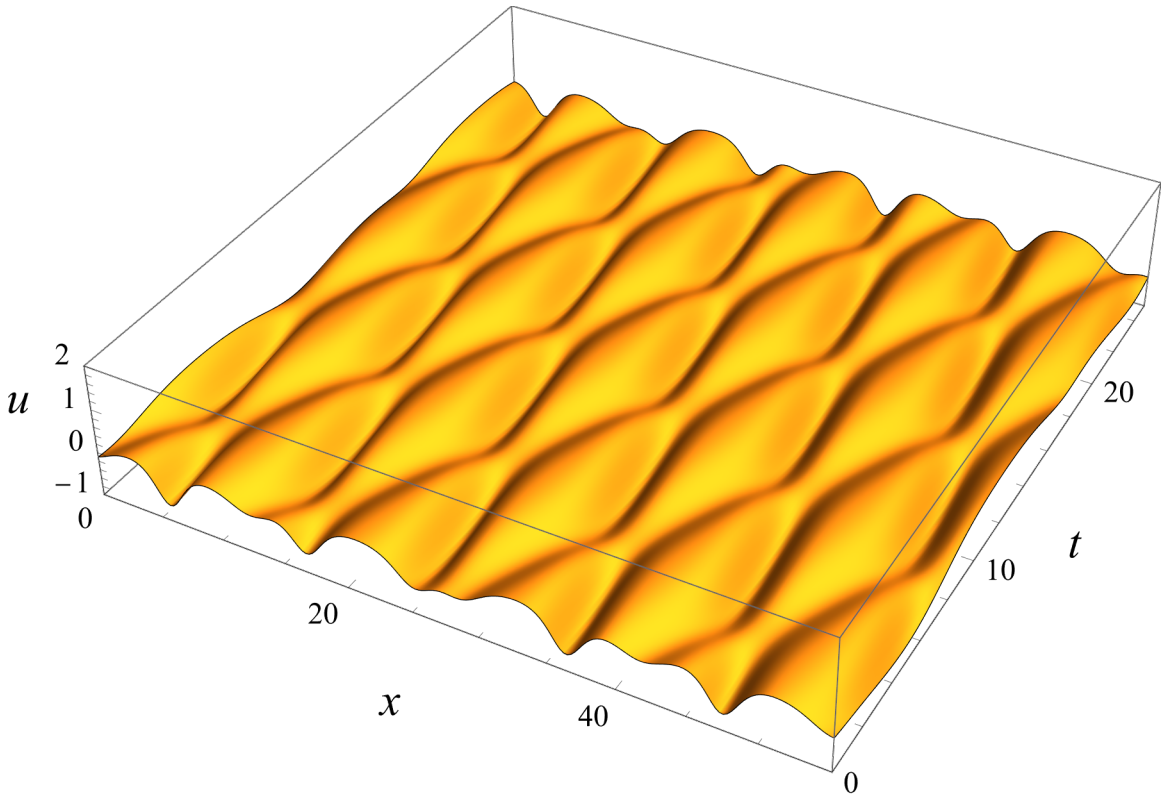
In[61]:=
```
Max[Abs[nstx2 - Transpose[nsxt2]]]
```

Out[61]=
```
0.000921854
```

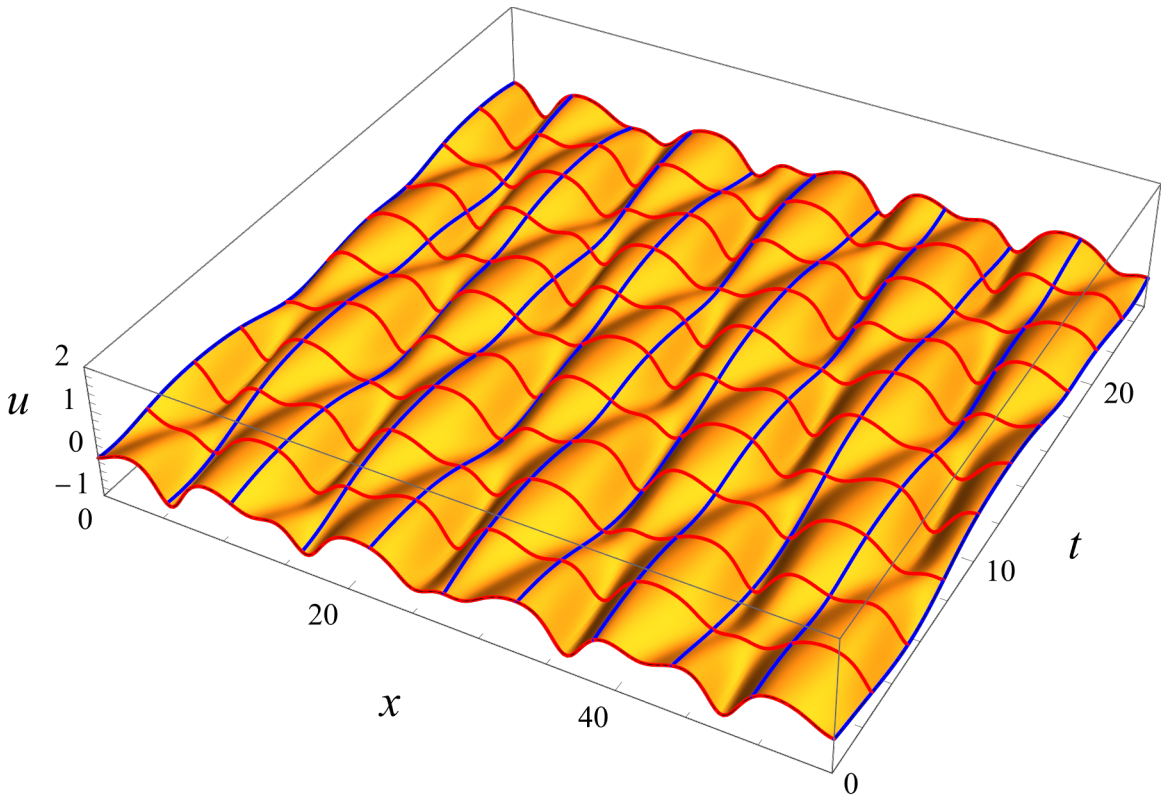
Теперь можно использовать полученный массив точек (любой из двух, поскольку они совпадают), чтобы построить график решения.

In[62]:=
```
gr3 = ListPlot3D[nstx2,
  DataRange → {{x0, x1}, {t0, t1}},
  PlotRange → {-1, 2},
  Mesh → None,
  AxesLabel → Evaluate[Style[#, Italic, 24] & /@ {"x", "t", "u"}],
  ImageSize → 600,
  BaseStyle → {FontFamily → "Times", FontSize → 16},
  BoxRatios → {1, 1, 0.2},
  PlotRangePadding → 0]

gr2 = Show[gr3, gr1]
```

Out[62]=



Out[63]=



3 Тождества для уравнений Дубровина

Символьная проверка (при $n = 1, 2, 3$) того, что:

1) определенные на лекции векторные поля по x и t коммутируют;

2) переменная u, построенная по совместному решению соответствующих ОДУ, удовлетворяет КдФ;
3) интегралы от голоморфных дифференциалов служат первыми интегралами (неавтономными).

Вводим основные выражения:

$P(y) = (y - e_1) \ldots (y - e_{2\,n+1}), \qquad \Pi_j = \prod_{i \neq j}(y_j - y_1), \qquad Y = y_1 + \ldots + y_n, \quad 2\,E = e_1 + \ldots + e_{2\,n+1}, \quad 2\,u = Y - E$

In[64]:=
```
P[y_, n_] := Product[y - e[i], {i, 1, 2 n + 1}]
prod[j_, n_] := Product[If[j == s, 1, y[j] - y[s]], {s, 1, n}]
se[n_] := 1/2 Sum[e[i], {i, 1, 2 n + 1}]
sy[n_] := Sum[y[j], {j, 1, n}]
u[n_] := (sy[n] - se[n])/2
```

Определяем дифференцирования по x и t

$$y_{j,x} = \frac{\sqrt{P(y_j)}}{\Pi_j}, \quad y_{j,t} = \left(E - Y + y_j\right)\frac{\sqrt{P(y_j)}}{\Pi_j}$$

In[69]:=
```
ddx[f_, n_] := Sum[Sqrt[P[y[j], n]]/prod[j, n] D[f, y[j]], {j, 1, n}]
ddx[f_, n_, k_] := Nest[ddx[#, n] &, f, k]

ddt[f_, n_] := Sum[(se[n] - sy[n] + y[j]) Sqrt[P[y[j], n]]/prod[j, n] D[f, y[j]], {j, 1, n}]
```

Проверяем их коммутативность на векторе из динамических переменных

In[72]:=
```
yy[n_] := Table[y[j], {j, 1, n}]

Table[Together[ddt[ddx[yy[n], n], n] - ddx[ddt[yy[n], n], n]], {n, 1, 3}]
```

Out[73]=
```
{{0}, {0, 0}, {0, 0, 0}}
```

Проверка того, что u удовлетворяет КдФ (для $n = 3$ требует минуты счета)

In[74]:=
```
Table[Together[ddt[u[n], n] - ddx[u[n], n, 3] + 6 u[n] ddx[u[n], n]], {n, 1, 2}]
```

Out[74]=
```
{0, 0}
```

In[75]:=
```
(* Timing[Together[ddt[u[3],3]-ddx[u[3],3,3]+6u[3]ddx[u[3],3]]] *)
```

Первые интегралы

$$F_k = \sum_{j=1}^{n} \int_{y_0}^{y_j} \frac{y^k}{\sqrt{P(y)}}\, dy$$

Определяем не сами интегралы, а их градиенты, тогда производные по x и t получаются скалярным умножением на соответствующее векторное поле.

In[76]:=
```
gradF[k_, n_] := Table[y[j]^k/Sqrt[P[y[j], n]], {j, 1, n}]
```

$n = 1$. Производные по x и t – постоянные, то есть F_0 – линейная функция от x, t

In[77]:=
```
Together[gradF[0, 1].ddx[yy[1], 1]]

Together[gradF[0, 1].ddt[yy[1], 1]]
```

Out[77]=
```
1
```

Out[78]=
```
1/2 (e[1] + e[2] + e[3])
```

$n = 2$. Производные по x и t – нули или постоянные, то есть F_0, F_1 линейны по x, t

In[79]:=
```
Together[gradF[0, 2].ddx[yy[2], 2]]
Together[gradF[1, 2].ddx[yy[2], 2]]

Together[gradF[0, 2].ddt[yy[2], 2]]
Together[gradF[1, 2].ddt[yy[2], 2]]
```

Out[79]=
```
0
```

Out[80]=
```
1
```

Out[81]=
```
1
```

Out[82]=
```
1/2 (e[1] + e[2] + e[3] + e[4] + e[5])
```

$n = 3$. То же для F_0, F_1, F_2

In[83]:=
```
Together[gradF[0, 3].ddx[yy[3], 3]]
Together[gradF[1, 3].ddx[yy[3], 3]]
Together[gradF[2, 3].ddx[yy[3], 3]]

Together[gradF[0, 3].ddt[yy[3], 3]]
Together[gradF[1, 3].ddt[yy[3], 3]]
Together[gradF[2, 3].ddt[yy[3], 3]]
```

Out[83]=
```
0
```

Out[84]=
```
0
```

Out[85]=
```
1
```

Out[86]=
```
0
```

Out[87]=
```
1
```

Out[88]=
```
1/2 (e[1] + e[2] + e[3] + e[4] + e[5] + e[6] + e[7])
```

$n = 4$. То же для F_0, F_1, F_2, F_3

In[89]:=
```
Together[gradF[0, 4].ddx[yy[4], 4]]
Together[gradF[1, 4].ddx[yy[4], 4]]
Together[gradF[2, 4].ddx[yy[4], 4]]
Together[gradF[3, 4].ddx[yy[4], 4]]

Together[gradF[0, 4].ddt[yy[4], 4]]
Together[gradF[1, 4].ddt[yy[4], 4]]
Together[gradF[2, 4].ddt[yy[4], 4]]
Together[gradF[3, 4].ddt[yy[4], 4]]
```

Out[89]=
$$0$$

Out[90]=
$$0$$

Out[91]=
$$0$$

Out[92]=
$$1$$

Out[93]=
$$0$$

Out[94]=
$$0$$

Out[95]=
$$1$$

Out[96]=
$$\frac{1}{2}\,(e[1]+e[2]+e[3]+e[4]+e[5]+e[6]+e[7]+e[8]+e[9])$$