

2022. Лекция 9, конечнозонные решения КдФ

1 Кноидальная волна

Решения КдФ в виде бегущей волны приводят к уравнению

$$u_x^2 = P(u) = 2u^3 - cu^2 + c_1u + c_2.$$

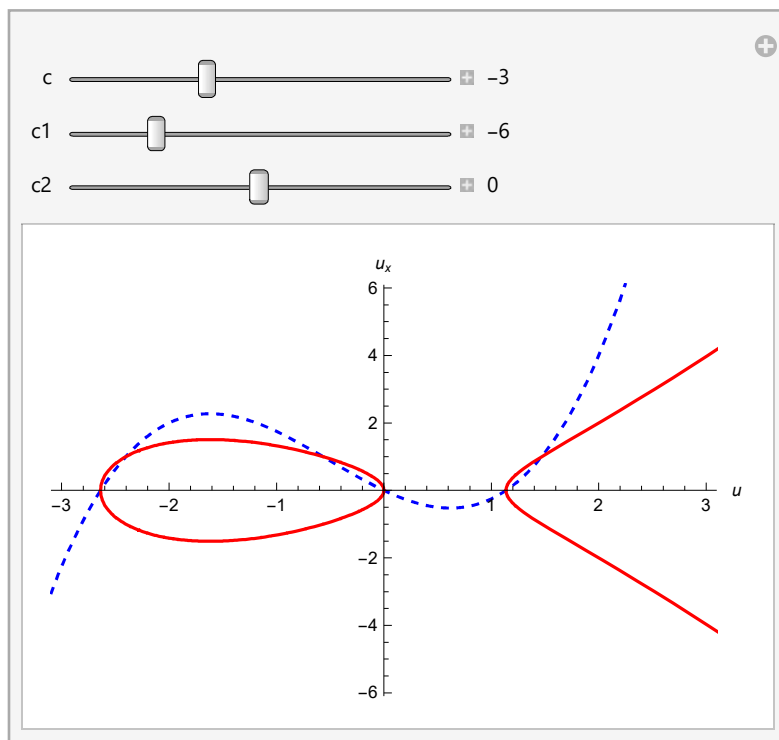
Оно определяет кривую в фазовой плоскости (u, u_x) . Форма кривой определяется нулями правой части. Если они вещественны и различны, то имеем овал и неограниченную ветвь. Если один корень вещественный и два комплексно сопряженных, то есть только одна неограниченная ветвь. На следующем графике можно посмотреть,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

In[1]:=

```
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"}
]
```

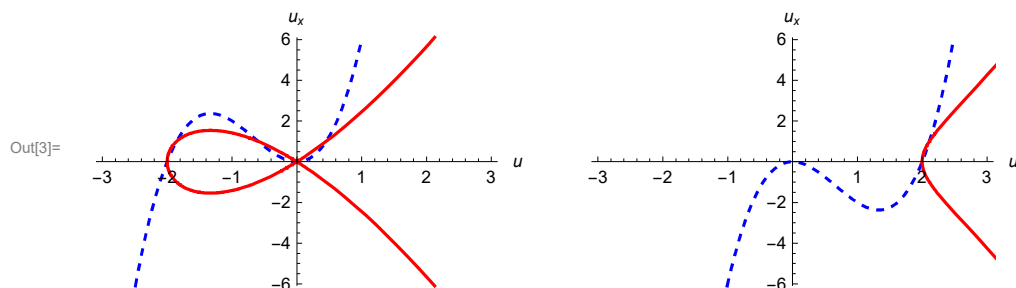
Out[2]=



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[3]:=

```
GraphicsRow[{plot[2 u^2 (u + 2)], plot[2 u^2 (u - 2)]}, ImageSize -> 500]
```



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

$$\begin{aligned} \operatorname{sn}' x &= \operatorname{dn} x \operatorname{cn} x, \\ \operatorname{cn}' x &= -\operatorname{dn} x \operatorname{sn} x, \\ \operatorname{dn}' x &= -k \operatorname{sn} x \operatorname{cn} x, \\ \operatorname{sn} 0 &= 0, \quad \operatorname{cn} 0 = \operatorname{dn} 0 = 1, \end{aligned}$$

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

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

В пределе $k \rightarrow 0$ получаем обычную тригонометрию. Предел $k \rightarrow 1$ отвечает, как мы увидим, солитону.

```

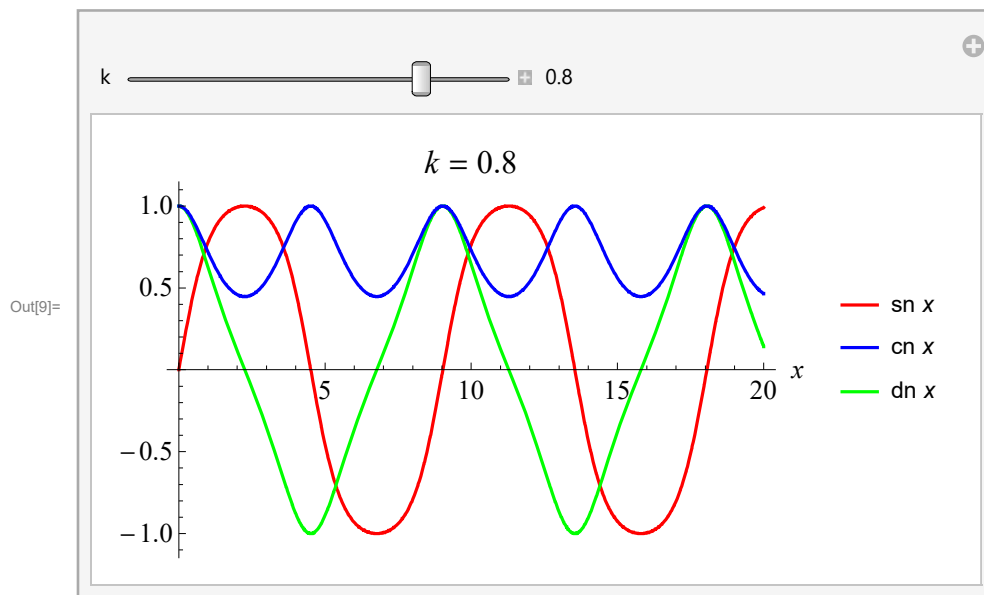
In[4]:= cn = JacobiCN[x, k];
sn = JacobiSN[x, k];
dn = JacobiDN[x, k];
{D[sn, x] - dn cn, D[cn, x] + dn sn, D[dn, x] + k sn cn,
 D[sn2 + cn2, x], D[k sn2 + dn2, x]}

plot1[kk_] := Plot[Evaluate[{sn, cn, dn} /. k → kk], {x, 0, 20},
 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[7]= {0, 0, 0, 0, 0}



```

In[10]:= Export[NotebookDirectory[] <> "sncndn.pdf", plot1[0.8], "PDF"];

```

Нетрудно показать, что если положить $u = K_1 + K_2 \text{sn}^2(bx)$, то u_x^2 будет представляться в виде многочлена третьей 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 (c1 - 2 c K1 + 6 K1^2 + 4 b^2 K2 - 2 c K2 + 12 K1 K2 + 6 K2^2) JacobiSN[d + b (-c t + x), k]^2 -
K2^2 (4 b^2 - c + 4 b^2 k + 6 K1 + 6 K2) JacobiSN[d + b (-c t + x), k]^4 +
2 K2^2 (2 b^2 k + K2) JacobiSN[d + b (-c t + x), k]^6
```

```
Out[15]= {{K1 -> 1/6 (-4 b^2 + c + 8 b^2 k), K2 -> -2 b^2 k, c1 -> 1/6 (-16 b^4 + c^2 + 16 b^4 k - 16 b^4 k^2),
c2 -> 1/108 (-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)}}}
```

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]
```

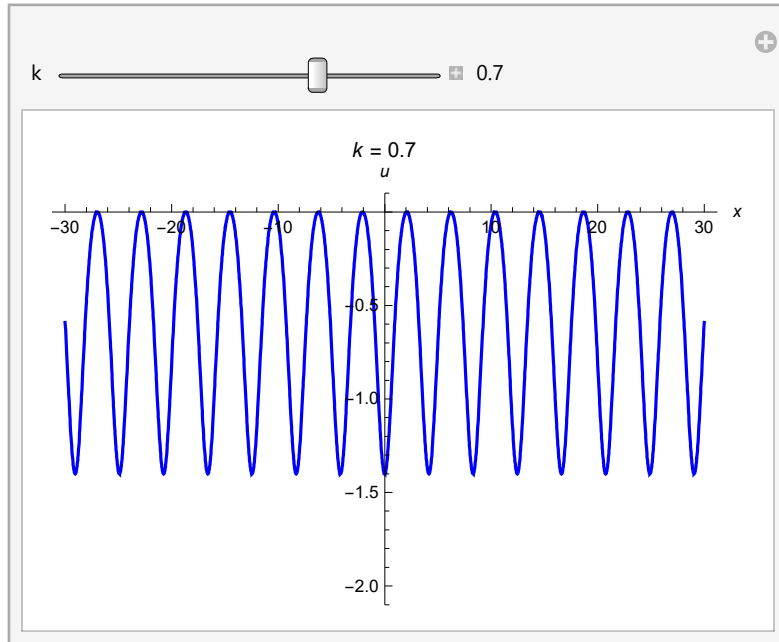
```
Out[17]= 0
```

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]:=

```
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"]]
```

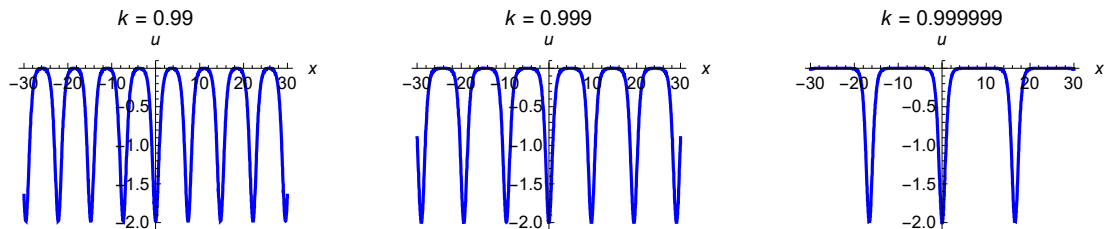
Out[19]=



In[20]:=

```
gg = GraphicsRow[{plot2[0.99], plot2[0.999], plot2[0.999999]}, ImageSize -> 600]
```

Out[20]=



In[21]:=

```
Export[NotebookDirectory[] <> "cnoidal.pdf", gg, "PDF"];
```

2 Двухзонное решение КдФ

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

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

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

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

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

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

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

```
In[22]:=
d[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]:=
dx[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]]
```

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 выше при определении D_t , сейчас лишь заменяем D_x на укороченное векторное поле, действующее на четырех переменных. Имеем:

```
In[30]:=
dt[f_] := ut D[f, u[0]] + dx[ut] D[f, u[1]] + dx[ut, 2] D[f, u[2]] + dx[ut, 3] D[f, u[3]]
```

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

```
In[31]:=
dt[u[0]]
Collect[dt[u[1]], {u[3], u[2], u[1], u[0]}, Factor]
Collect[dt[u[2]], {u[3], u[2], u[1], u[0]}, Factor]
Collect[dt[u[3]], {u[3], u[2], u[1], u[0]}, Factor]
```

Out[31]= -6 u[0] u[1] + u[3]

Out[32]= -c3 - c2 u[0] + 3 c1 u[0]^2 - 10 u[0]^3 - u[1]^2 + (-c1 + 4 u[0]) u[2]

Out[33]= (-c2 + 6 c1 u[0] - 30 u[0]^2) u[1] + 2 u[1] u[2] + (-c1 + 4 u[0]) u[3]

Out[34]= 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[35]:= Table[Expand[dx[dt[u[k]]] - dt[dx[u[k]]]], {k, 0, 3}]
```

```
Out[35]= {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[36]:= (* портим ut *)
ut = u[3] - 66 u[0] u[1];
Table[Expand[dx[dt[u[k]]] - dt[dx[u[k]]]], {k, 0, 3}]

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

```
Out[37]= {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[39]= {0, 0, 0, 0}
```

Переходим к численному счёту. Определяем решение задачи Коши по x .

```
In[40]:= 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] ==
    10 u[0][x] u[2][x] + 5 u[1][x]^2 - 10 u[0][x]^3 - c1 (u[2][x] - 3 u[0][x]^2) - c2 u[0][x] - 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]]
```

Определяем решение задачи Коши по t . Нам нужно добавить аргумент t к тем обозначениям, что у нас были. Сделаем это в следующей ячейке, а потом скопируем в аргументы NDSolve. (Конечно, такое руководство не есть хорошо. Для более серьезных задач нужно делать по другому, чтобы все было автоматически.)

```
In[41]:= dt[u[0]] /. u[k_] => u[k][t]
Expand[dt[u[1]]] /. u[k_] => u[k][t]
Expand[dt[u[2]]] /. u[k_] => u[k][t]
Expand[dt[u[3]]] /. u[k_] => u[k][t]
```

```
Out[41]= -6 u[0][t] u[1][t] + u[3][t]
```

```
Out[42]= -c3 - c2 u[0][t] + 3 c1 u[0][t]^2 - 10 u[0][t]^3 - u[1][t]^2 - c1 u[2][t] + 4 u[0][t] u[2][t]
```

```
Out[43]= -c2 u[1][t] + 6 c1 u[0][t] u[1][t] - 30 u[0][t]^2 u[1][t] +
2 u[1][t] u[2][t] - c1 u[3][t] + 4 u[0][t] u[3][t]
```

```
Out[44]= c1 c3 + c1 c2 u[0][t] - 4 c3 u[0][t] - 3 c1^2 u[0][t]^2 - 4 c2 u[0][t]^2 + 22 c1 u[0][t]^3 -
40 u[0][t]^4 + c1 u[1][t]^2 - 40 u[0][t] u[1][t]^2 + c1^2 u[2][t] - c2 u[2][t] -
8 c1 u[0][t] u[2][t] + 10 u[0][t]^2 u[2][t] + 2 u[2][t]^2 + 6 u[1][t] u[3][t]
```

```
In[45]:= nsolt[{u00_, u10_, u20_, u30_}, {c1_, c2_, c3_}, {t0_, t1_}] :=
NDSolve[
{u[0]'[t] == -6 u[0][t] u[1][t] + u[3][t],
u[1]'[t] ==
-c3 - c2 u[0][t] + 3 c1 u[0][t]^2 - 10 u[0][t]^3 - u[1][t]^2 - c1 u[2][t] + 4 u[0][t] u[2][t],
u[2]'[t] == -c2 u[1][t] + 6 c1 u[0][t] u[1][t] - 30 u[0][t]^2 u[1][t] +
2 u[1][t] u[2][t] - c1 u[3][t] + 4 u[0][t] u[3][t],
u[3]'[t] == c1 c3 + c1 c2 u[0][t] - 4 c3 u[0][t] - 3 c1^2 u[0][t]^2 - 4 c2 u[0][t]^2 +
22 c1 u[0][t]^3 - 40 u[0][t]^4 + c1 u[1][t]^2 - 40 u[0][t] u[1][t]^2 + c1^2 u[2][t] -
c2 u[2][t] - 8 c1 u[0][t] u[2][t] + 10 u[0][t]^2 u[2][t] + 2 u[2][t]^2 + 6 u[1][t] u[3][t],
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[46]:= 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[54]:=

```
(* 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}];

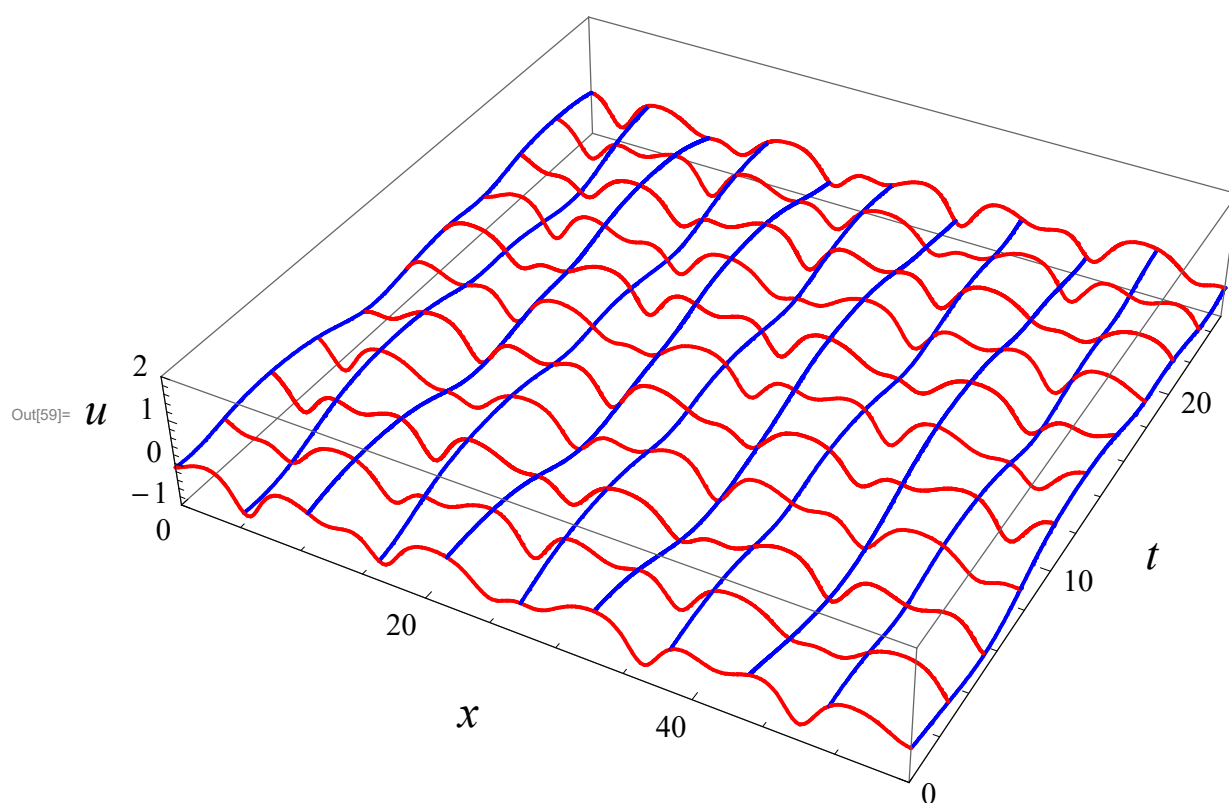
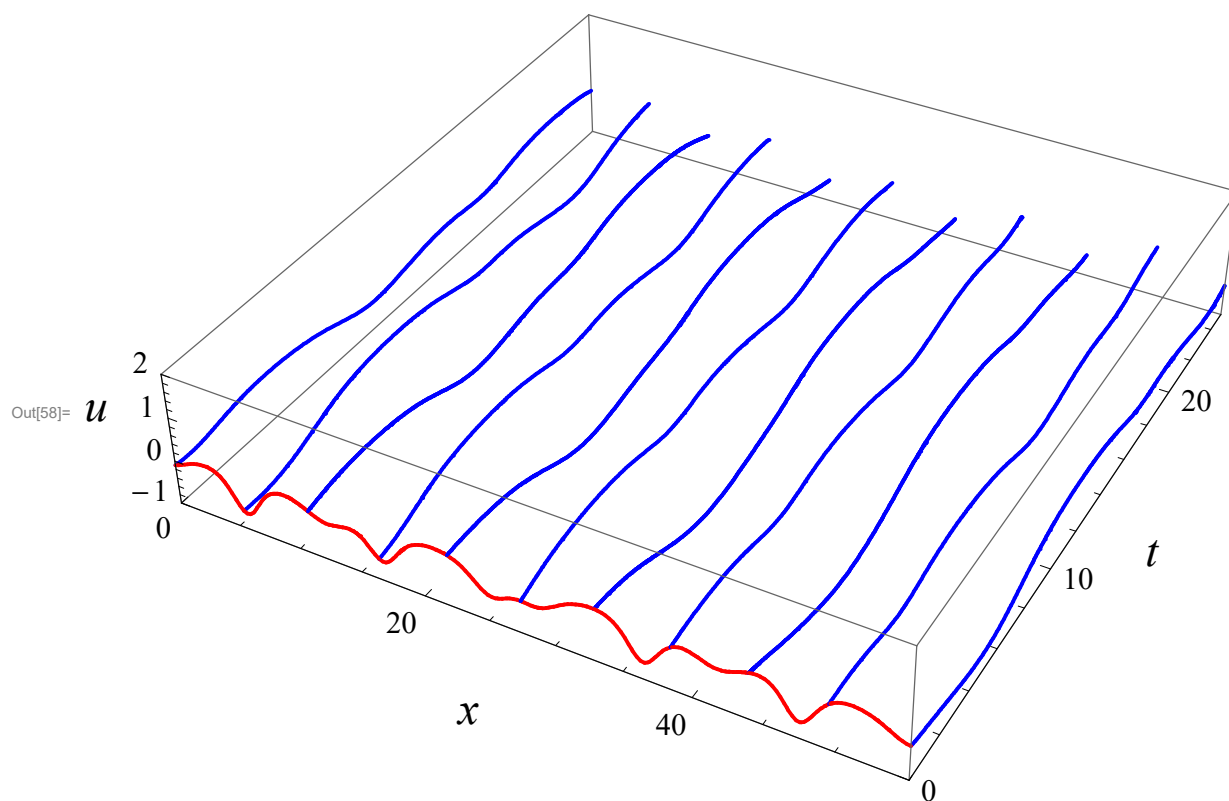
nstx1 = Table[
  ns = nsolx[
    {u[0][t], u[1][t], u[2][t], u[3][t]} /. t -> t0 + j (t1 - t0) / mt /. nst, cc, {x0, x1}];
  Line[Table[{x0 + i (x1 - x0) / Mx, t0 + j (t1 - t0) / mt, u[0][x0 + i (x1 - x0) / Mx] /. ns},
    {i, 0, Mx}]],
  {j, 0, mt}];

(* 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}]
```



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[60]:= (* 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[64]:= Max[Abs[nstx2 - Transpose[nsxt2]]]

```

```

Out[64]= 0.00092204

```

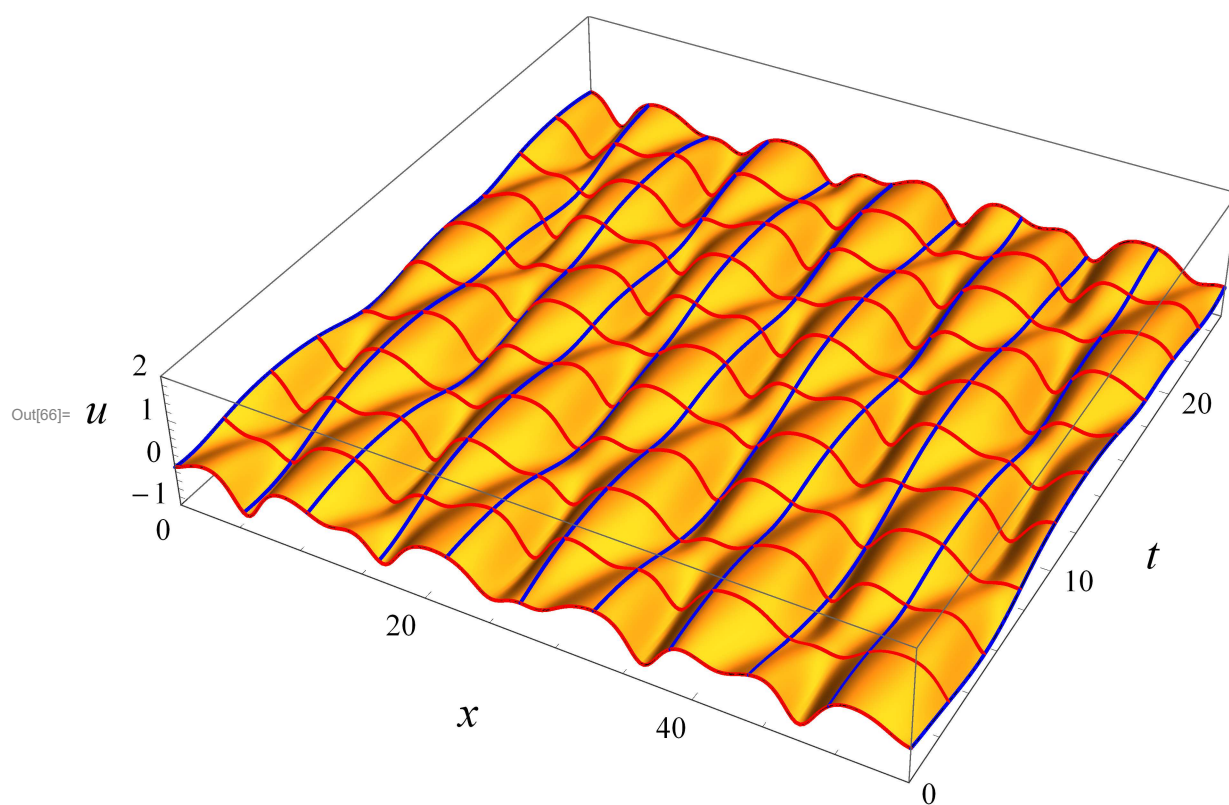
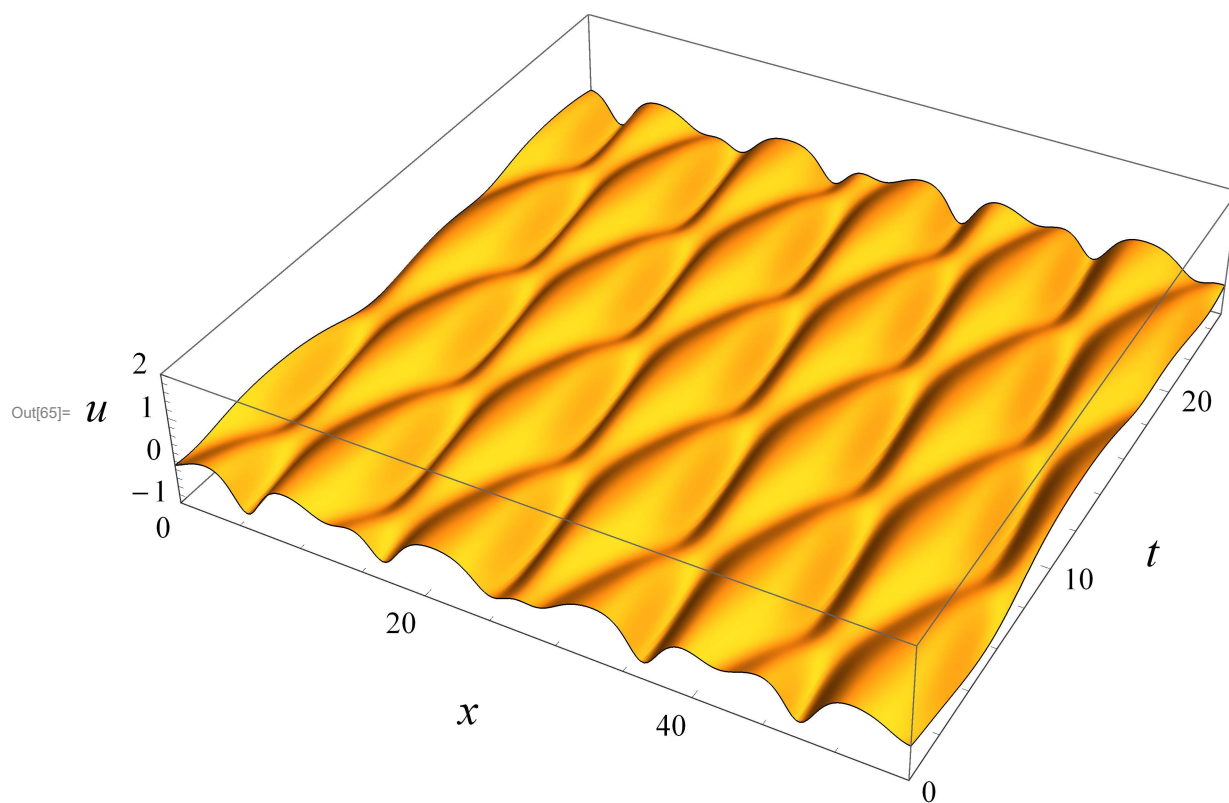
Теперь можно использовать полученный массив точек (любой из двух), чтобы построить график решения.

```

In[65]:= 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]

```



Сохраняем графики и вставляем в лекцию.

In[67]:=

```
Export[NotebookDirectory[] <> "L9pic0.pdf", gr0, "PDF"];  
Export[NotebookDirectory[] <> "L9pic1.pdf", gr1, "PDF"];  
Export[NotebookDirectory[] <> "L9pic2.pdf", gr2, "PDF"];  
Export[NotebookDirectory[] <> "L9pic3.pdf", gr3, "PDF"];
```