- A very slightly different version of this paper has been presented at the Wolfram Technology Conference 2005 in Champaign and is available through http://library.wolfram.com/infocenter/Conferences/5790/ – Y.Papegay – Editor.

Event location at integration of ODEs with jumping nonlinearity

Jirí Benedikt

Department of Mathematics
University of West Bohemia
30614 Plzec
Czech Republic

benedikt@kma.zcu.cz

The author has been supported by the Grant 1N04078 and by the Research Plan MSM 4977751301 of the Ministry of Education, Youth and Sports of the Czech Republic.

Abstract

We are interested in the so–called *Fučík spectrum* of the *fourth−order Dirichlet boundary value problem*

$$u^{(4)}(t) = \mu u^{+}(t) - \nu u^{-}(t), \qquad t \in [0, 1],$$
$$u(0) = u'(0) = u(1) = u'(1) = 0,$$

where $u^{+} = \max\{u, 0\}$ and $u^{-} = \max\{-u, 0\}$, i.e. $u = u^{+} - u^{-}$. The Fučík spectrum is defined as the set of all couples $(\mu,\nu) \in R^2$ such that the problem has a nontrivial solution, and it can be figured by *Mathematica* as the zero–contour of a function $D = D(\mu, \nu)$. Existence of a stable solution of an initial–boundary value problem that describes the oscillation of a suspension bridge is related to the shape of the Fučík spectrum of our problem.

Introduction

Mathematical model of a suspension bridge

There are many mathematical models of suspension bridge, at different levels of simplification. One of the most simplified models is the fourth−order Dirichlet boundary value problem

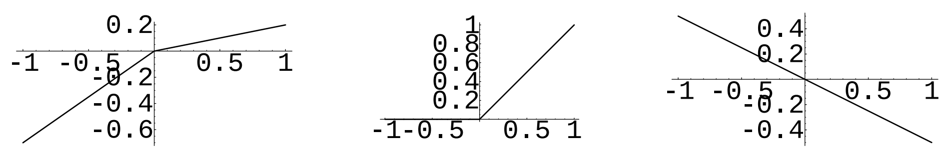
$$u^{(4)}(t) = f(u(t)), \qquad t \in [0, 1],$$
$$u(0) = u'(0) = u(1) = u'(1) = 0,$$

where $u : [0, 1] \to \boldsymbol{R}$ (the solution) stands for the displacement of the bridge deck and $f : \boldsymbol{R} \to \boldsymbol{R}$ describes the external force. The boundary conditions are called Dirichlet and mean that the ends of the bridge deck are clamped.

The external force f depends on the displacement u, as it is typical for the suspension bridge – if the bridge deck moves downwards (i.e. $u > 0$ – we consider the downwards direction to be positive), the cable stays pull it back, if it moves upwards ($u < 0$), the cable stays do nothing.

Let us assume that f depends linearly on u, but with different constants for u positive and u negative, i.e. $f = \mu u^+ - \nu u^-$, where $u^+ = \max\{u, 0\}$ is the positive and $u^- = \max\{-u, 0\}$ the negative part of u (hence, $u = u^+ - u^-$). This is satisfied, if the cable stays behave like a spring with the modulus μ for u positive and the modulus ν for u negative ($\nu = 0$ in our case). Note that such f is called a *jumping nonlinearity*.

```
In[1]:= PosPart[a_] := (1 / 2) * (a + Abs[a]);
        NegPart[a_] := (1 / 2) * (Abs[a] - a);
        Show[GraphicsArray[
           (Plot[#1[[1]] * PosPart[t] - #1[[2]] * NegPart[t], {t, -1, 1},
               AspectRatio → Automatic, DisplayFunction → Identity] &) /@
            {{0.2, 0.7}, {1, 0}, {-0.5, -0.5}}]];
```



□ The Fučík spectrum

We arrive at the fourth–order Dirichlet boundary value generalized eigenvalue problem

$$u^{(4)}(t) = \mu u^+(t) - \nu u^-(t), \qquad t \in [0, 1],$$
$$u(0) = u'(0) = u(1) = u'(1) = 0.$$

The Fučík (generalized) spectrum is defined as the set of all couples $(\mu,\nu) \in \boldsymbol{R}^2$ such that the problem has a nontrivial solution (*eigenfunction*).

If f is at least *asymptotically equal* to $\mu u^+ - \nu u^-$, i.e.

$$\lim_{s \to +\infty} \frac{f(s)}{s} = \mu \quad \text{and} \quad \lim_{s \to -\infty} \frac{f(s)}{s} = \nu,$$

and (μ, ν) does not belong to the Fučík spectrum, then the topological degree theory can be used to prove that the solution is a priori bounded (otherwise there may exist an unbounded sequence of solutions). Hence it is important to know the Fučík spectrum well. Our aim is to **figure the Fučík spectrum** by *Mathematica* (we use the version 5.1.1.0).

■ Shooting method

□ Initial value problem

Let us consider the fourth–order *initial value problem* (IVP)

$$u^{(4)}(t) = \mu u^{+}(t) - \nu u^{-}(t), \qquad t \in [0, 1],$$
$$u(0) = u'(0) = 0, \; u''(0) = 1, \; u'''(0) = \delta,$$

where $\mu, \nu > 0$ and $\delta \in \boldsymbol{R}$. The solution can be proved to be unique, and so we may denote by $V = V(\mu, \nu, \delta)$ the value of the solution at $t = 1$. Obviously, V is an increasing continuous function of δ, and $\lim_{\delta \to \pm\infty} V(\mu, \nu, \delta) = \pm\infty$. Hence there exists exactly one constant $\delta_0 = \delta_0(\mu, \nu)$, such that $V(\mu, \nu, \delta_0(\mu, \nu)) = 0$. Since clearly $V(\mu, \nu, \delta) > 0$ for any $\delta \geq 0$, it must be $\delta_0(\mu, \nu) < 0$ for all $\mu, \nu > 0$. Let $D = D(\mu, \nu)$ denote the first derivative of the solution of the IVP with $\delta = \delta_0(\mu, \nu)$. If $D(\mu, \nu) = 0$, then (μ, ν) belongs to the Fučík spectrum. Moreover, (ν, μ) belongs to the Fučík spectrum, too, and the corresponding eigenfunction is $-u$.

On the other hand, let us assume that (μ, ν), $\mu, \nu > 0$, belongs to the Fučík spectrum and let u be a corresponding eigenfunction. It can be proved that $u''(0) \neq 0$. Let us assume first $u''(0) > 0$. Since αu, $\alpha > 0$, is an eigenfunction, too, we may suppose that $u''(0) = 1$. Then, clearly, u is the solution of the IVP with $\delta = \delta_0(\mu, \nu)$, and so $D(\mu, \nu) = 0$. If $u''(0) < 0$, then $-u$ is an eigenfunction, corresponding to (ν, μ) with $(-u)''(0) > 0$. Consequently, $D(\nu, \mu) = 0$ in this case.

We showed that (μ, ν), $\mu, \nu > 0$, belongs to the Fučík spectrum, **if and only if** $D(\mu, \nu) = 0$ or $D(\nu, \mu) = 0$. Hence it may be figured as a union of the zero contours of $D(\mu, \nu)$ and $D(\nu, \mu)$. Our aim now reduces to implementation of D.

■ Implementation

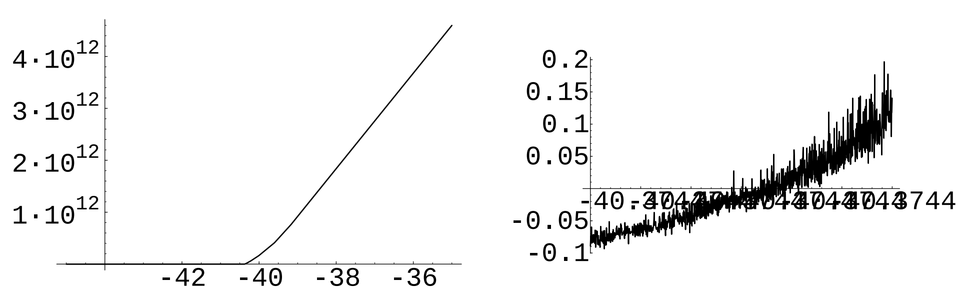
□ The first problem: precision

From now on we write μ^4 and ν^4 instead of μ and ν in the differential equation. First we implement the function V. The simplest one might be

In[4]:=
```
ValueAtOne[μ_, ν_, δ_] :=
  u[1] /. NDSolve[{Derivative[4][u][t] == μ^4 * PosPart[u[t]] -
      ν^4 * NegPart[u[t]], u[0] == 0,
      Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
      Derivative[3][u][0] == δ}, u, {t, 0, 1}][[1]];
```

Now we look for $\delta_0(\mu, \nu)$ as a root of $V(\mu, \nu, \delta)$.

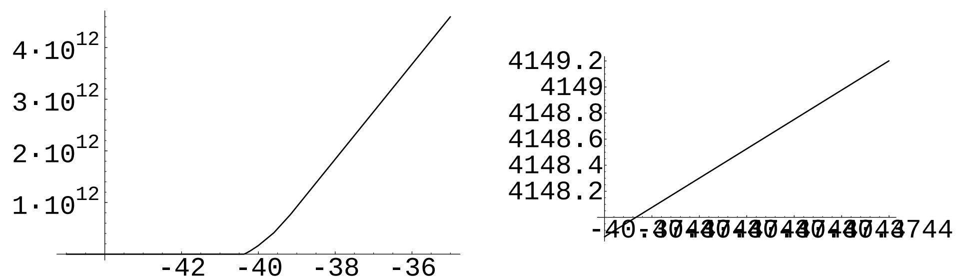
In[5]:=
```
p1 = Plot[ValueAtOne[40, 30, t],
    {t, -45, -35}, DisplayFunction → Identity];
p2 = Plot[ValueAtOne[40, 30, t], {t, -40.37441848174,
     -40.37441848177}, DisplayFunction → Identity];
Show[GraphicsArray[{p1, p2}]];
```



We see that if we want to guarantee, e.g., $\left|V(\mu, \nu, \delta_0(\mu, \nu))\right| < 10^{-6}$, then the "machine precision" is not enough. So let us improve the implementation of V.

In[8]:=
```
ValueAtOne[μ_, ν_, δ_, goal_, working_] :=
  u[1] /. NDSolve[{Derivative[4][u][t] ==
       SetPrecision[μ^4, working] * PosPart[u[t]] -
        SetPrecision[ν^4, working] * NegPart[u[t]], u[0] == 0,
      Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
      Derivative[3][u][0] == SetPrecision[δ, working]},
     u, {t, 0, 1}, PrecisionGoal → goal,
     WorkingPrecision → working][[1]];
```

In[9]:=
```
p1 = Plot[ValueAtOne[40, 30, t, 20, 40],
   {t, -45, -35}, DisplayFunction → Identity];
p2 = Plot[ValueAtOne[40, 30, t, 20, 40], {t, -40.37441848174,
    -40.37441848177}, DisplayFunction → Identity];
Show[GraphicsArray[{p1, p2}]];
```



Increasing of the working precision caused a significant change of the obtained values. To find an optimal precision we can compare the values for different precisions.

In[12]:=
```
TableForm[Table[ValueAtOne[40, 30, -40.37441848177, n, 2 * n],
  {n, 15, 25}], TableHeadings →
  {Table[2 * n, {n, 15, 25}], {"working", "result"}}]
```

```
NDSolve::mxst :
 Maximum number of 10000 steps reached at the point t ==
   0.158915875742668323463501024550776480465923783645. More…
```

```
InterpolatingFunction::dmval :
 Input value {1} lies outside the range of data in the
   interpolating function. Extrapolation will be used. More…
```

Out[12]//TableForm=

30	4147.844009112627879036194219
32	4147.85020676900216790826638637
34	4147.8507159791465763152859015341
36	4147.850720342217179176002726697527
38	4147.85072103731465664076884363472049
40	4147.8507211735240502352115365517091080
42	4147.850721189333104132293720867813757838
44	4147.85072119207624366125682054932031268129
46	4147.8507211926321593247874215881657174450701
48	$2.01380531575 \times 10^{26}$
50	4147.85072119265397022706925474453125951484798668

Obviously, **`PrecisionGoal`** 20 and **`WorkingPrecision`** 40 is enough if we want the (absolute) precision of the result to be less than 10^{-6}. But what happened when the **`WorkingPrecision`** was 48!? Let us plot the solution for this case.

In[13]:=
```
sol =
  u /. NDSolve[{Derivative[4][u][t] == SetPrecision[40^4, 48] *
         PosPart[u[t]] - SetPrecision[30^4, 48] * NegPart[u[t]],
      u[0] == 0, Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
      Derivative[3][u][0] == SetPrecision[-40.37441848177, 48]},
     u, {t, 0, 1}, PrecisionGoal → 24, WorkingPrecision → 48][[1]]
Plot[sol[t], {t, 0, 1}, PlotRange → {{0, 1}, {-0.01, 0.01}}];
```

```
NDSolve::mxst :
 Maximum number of 10000 steps reached at the point t ==
   0.158915875742668323463501024550776480465923783645. More…
```

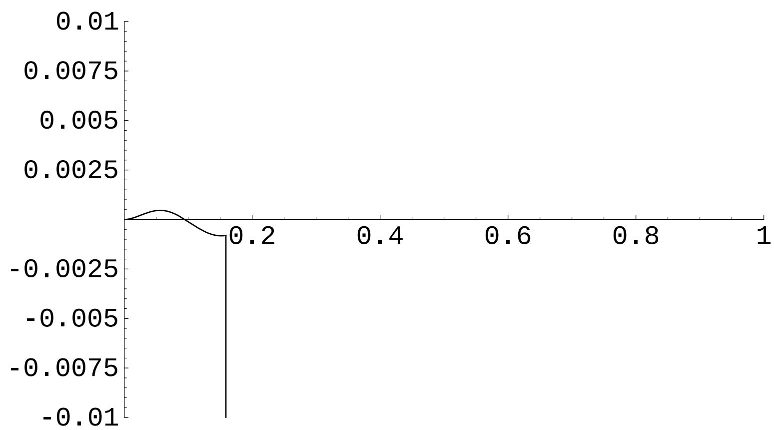
Out[13]=
```
InterpolatingFunction[
 {{0, 0.158915875742668323463501024550776480465923783645}},
 <>]
```

```
InterpolatingFunction::dmval :
 Input value {0.166892} lies outside the range of data in the
   interpolating function. Extrapolation will be used. More…
```

```
InterpolatingFunction::dmval :
 Input value {0.162122} lies outside the range of data in the
   interpolating function. Extrapolation will be used. More…
```

```
InterpolatingFunction::dmval :
 Input value {0.159579} lies outside the range of data in the
   interpolating function. Extrapolation will be used. More…
```

```
General::stop :
 Further output of InterpolatingFunction::dmval will be
   suppressed during this calculation. More…
```



The problem is that the solution was not computed on the whole interval [0, 1] and extrapolation was used. We can fix the problem using the **`StiffnessSwitching`** method.

In[15]:=
```
ValueAtOne[μ_, ν_, δ_, goal_, working_] :=
  u[1] /. NDSolve[{Derivative[4][u][t] ==
       SetPrecision[μ^4, working] * PosPart[u[t]] -
        SetPrecision[ν^4, working] * NegPart[u[t]], u[0] == 0,
      Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
      Derivative[3][u][0] == SetPrecision[δ, working]}, u,
     {t, 0, 1}, PrecisionGoal → goal, WorkingPrecision → working,
     Method → {StiffnessSwitching}][[1]];
```

In[16]:=
```
TableForm[Table[ValueAtOne[40, 30, -40.37441848177, n, 2*n],
  {n, 15, 25}], TableHeadings →
  {Table[2*n, {n, 15, 25}], {"working", "result"}}]
```

Out[16]//TableForm=

30	4147.850539168947270588051973
32	2354.624380592540479186154070999
34	4073.3797137633279322010284233788
36	4147.850720494023771567460056185661
38	4147.83375663037588578460793401693914
40	4147.8507029638944582979450700973703438
42	4144.690769953244960926007724027825195768
44	4147.61859372526944555661594669937064944536
46	4146.3270592512689254466365793133653519215778
48	4147.683873484625938621052746662330203035864920
50	4147.85072119265405482969749315775887465114041394

The equation was always solved on the whole [0, 1]. However, the results are obviously not consistent.

□ The second problem: jumping nonlinearity

The reason why we got such nonconsistent results is that our f (the right–hand side) is the jumping nonlinearity which is not differentiable at 0. It brings unpredictable errors into the solution as it changes its sign. Consequently, we must stop the integration whenever the solution attains the zero value, and restart the integration. This is easily accomplished with the `EventLocator` method, new in *Mathematica* 5.1.

In[17]:=
```
state = First[NDSolve`ProcessEquations[
    {Derivative[4][u][t] == SetPrecision[40^4, 130] *
        PosPart[u[t]] - SetPrecision[30^4, 130] * NegPart[u[t]],
     u[0] == 0, Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
     Derivative[3][u][0] == SetPrecision[-40.37441848177, 130]},
    u, t, Method → {EventLocator, "Event" → u[t],
      Method → StiffnessSwitching},
    WorkingPrecision → 130, AccuracyGoal → 30,
    PrecisionGoal → 30, MaxSteps → Infinity]];
result = Reap[While[Sow[state["CurrentTime"]["Forward"]] < 1,
    state = NDSolve`ReinitializeVector[
      state, state["CurrentTime"]["Forward"]],
      state["SolutionVector"]["Forward"]]];
    NDSolve`Iterate[state, 1];];
   state["SolutionVector"]["Forward"]]];
SetPrecision[result[[1]][[1]], 30]
```

Out[19]= 4147.85072119265408024709603968

The used precision appears to be sufficient for any $\mu, \nu \in [0, 50]$. Using the `EventLocator` is not appropriate when $\mu = \nu$, and so we finally define

In[20]:=
```
ValueAtOne[order_, μ_, ν_, δ_, goal_, working_] :=
  If[Abs[μ - ν] < 10^(-10), SetPrecision[
    Derivative[order][u][1] /. NDSolve[{Derivative[4][u][t] ==
          SetPrecision[μ^4, working] * PosPart[u[t]] -
           SetPrecision[ν^4, working] * NegPart[u[t]], u[0] == 0,
         Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
         Derivative[3][u][0] == SetPrecision[δ, working]},
        u, {t, 0, 1}, Method → StiffnessSwitching,
        WorkingPrecision → working, AccuracyGoal → goal,
        PrecisionGoal → goal, MaxSteps → Infinity][[1]][[1]],
    goal], Module[{state, result}, state =
     First[NDSolve`ProcessEquations[{Derivative[4][u][t] ==
         SetPrecision[μ^4, working] * PosPart[u[t]] -
          SetPrecision[ν^4, working] * NegPart[u[t]], u[0] == 0,
        Derivative[1][u][0] == 0, Derivative[2][u][0] == 1,
        Derivative[3][u][0] == SetPrecision[δ, working]},
       u, t, Method → {EventLocator, "Event" → u[t],
         Method → StiffnessSwitching},
       WorkingPrecision → working, AccuracyGoal → goal,
       PrecisionGoal → goal, MaxSteps → Infinity]];
    result = Reap[While[Sow[state["CurrentTime"]["Forward"]] <
        1, state = NDSolve`ReinitializeVector[
         state, state["CurrentTime"]["Forward"]],
         state["SolutionVector"]["Forward"]]];
       NDSolve`Iterate[state, 1];];
      state["SolutionVector"]["Forward"]]];
    SetPrecision[result[[1]][[order + 1]], goal]]];
```

In[21]:=
```
ValueAtOne[0, 40, 30, -40.37441848177, 30, 130] // Timing
```

Out[21]=
```
{2.764 Second, 4147.850721192654080247096039 68}
```

Having the function V implemented, we can easily compute $\delta_0(\mu, \nu)$ and $D(\mu, \nu)$.

In[22]:=
```
LookForNeg[fction_] := NestWhile[#1 * 2 &, -1, fction[#1] > 0 &];
InitInt[x_, prec_] :=
  SetPrecision[If[x == -1, {-1, 0}, {x, x / 2}], prec];
Halving[{l_, r_, f_}] :=
  If[f[(l + r) / 2] < 0, {(l + r) / 2, r, f}, {l, (l + r) / 2, f}];
MyFindRoot[fction_, goal_, working_] := NestWhile[Halving,
    Append[InitInt[LookForNeg[fction], working], fction],
    #1[[2]] - #1[[1]] > 10^(-goal) &][[1]];
DerZero[μ_, ν_] := ValueAtOne[1, μ, ν, MyFindRoot[
    ValueAtOne[0, μ, ν, #1, 30, 130] &, 25, 35], 30, 130];
```

In[27]:=
```
MyFindRoot[ValueAtOne[0, 40, 30, #, 30, 130] &, 25, 35]
DerZero[40, 30] // Timing
```

Out[27]=
```
-40.37441858031726698158182214056 8217
```

Out[28]=
```
{297.337 Second, -0.021774824527892359583448133 5221}
```

Finally, the Fučík spectrum may be figured by the `ContourPlot` function as follows

```
ContourPlot[DerZero[μ, ν]*DerZero[μ, ν],
  {μ, 0, 50}, {ν, 0, 50}, ContourShading -> None,
  Contours -> {0}, PlotPoints -> 200]
```

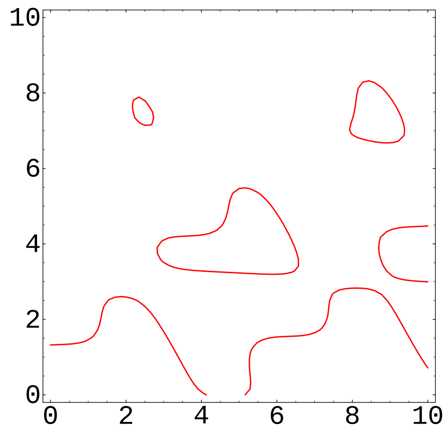
But it would take months...

■ Optimization

The **`ContourPlot`** method evaluates the function uniformly at all $n \times n$ points. We develop an adaptive iterative algorithm for figuring the zero contour of a function. It evaluates the function more densely near already found lines to make them smoother, but still it searches for another components of the contour.

Let us consider the following test function.

```
In[29]:= f[x_, y_] = Sin[x + Cos[x + y]] * Sin[y + 0.2] + x / 25 - y / 8 - 0.08;
         p = ContourPlot[f[x, y], {x, 0, 10},
            {y, 0, 10}, ContourShading → None, Contours → {0},
            ContourStyle → Hue[0], PlotPoints → 65];
```



To draw the picture, **`ContourPlot`** needed to evaluate the function at $65 \times 65 = 4225$ points. Our algorithm follows.

```
In[31]:= size = 10;
         proporref = .9;
         InsertNew[main_, pos_] := Module[{am, new, numnew}, am = main;
            new =
             Complement[pos, am[[2]] /. {x_?NumericQ, y_, f_} → {x, y}];
            numnew = Length[new];
            am = {am[[1]], Join[am[[2]], new]};
            am = am /. {x_?NumericQ, y_} → {x, y, f[x, y]};
            am = {am[[1]], Sort[am[[2]], #1[[2]] < #2[[2]] ||
                 #1[[2]] == #2[[2]] && #1[[1]] < #2[[1]] &]};
            {numnew, am}];
         Uni[main_] :=
           Module[{am, posnew, numnew}, am = MapAt[# + 1 &, main, {1, 1}];
            posnew = Flatten[Table[{j * size, i * size}, {i, 0, 1,
                2^-am[[1]][[1]]}, {j, 0, 1, 2^-am[[1]][[1]]}], 1];
            {numnew, am} = InsertNew[am, posnew];
            am = MapAt[
              # - ((2^am[[1]][[1]] + 1)^2 - (am[[1]][[3]] + numnew)) &,
              am, {1, 4}];
            ReplacePart[am, (2^am[[1]][[1]] + 1)^2, {1, 3}]];
         RefDiv[main_, lev_] :=
           Module[{am, posnew, numnew}, am = main;
            amchoice = Select[am[[2]],
              Mod[Take[#, 2], size * 2^-lev] == {0, 0} &];
            posnew = Select[amchoice,
                Module[{thenext}, thenext = Select[amchoice, Function[a,
                     Take[#, 2] + {size, 0} * 2^-(lev - 1) == Take[a, 2]], 1];
                  thenext =!= {} && #[[3]] * thenext[[1]][[3]] ≤ 0] &] /.
              {x_?NumberQ, y_, f_} → {x + size * 2^-lev, y};
            posnew = Join[posnew, Select[amchoice,
                 Module[{thenext}, thenext = Select[amchoice, Function[a,
                      Take[#, 2] + {0, size} * 2^-(lev - 1) == Take[a, 2]], 1];
                   thenext =!= {} && #[[3]] * thenext[[1]][[3]] ≤ 0] &] /.
               {x_?NumberQ, y_, f_} → {x, y + size * 2^-lev}];
            {numnew, am} = InsertNew[am, posnew];
            MapAt[# + numnew &, am, {1, 4}]];
         RefAdd[main_, lev_] :=
           Module[{am, posnew, numnew}, am = main;
            amchoice = Select[am[[2]],
```

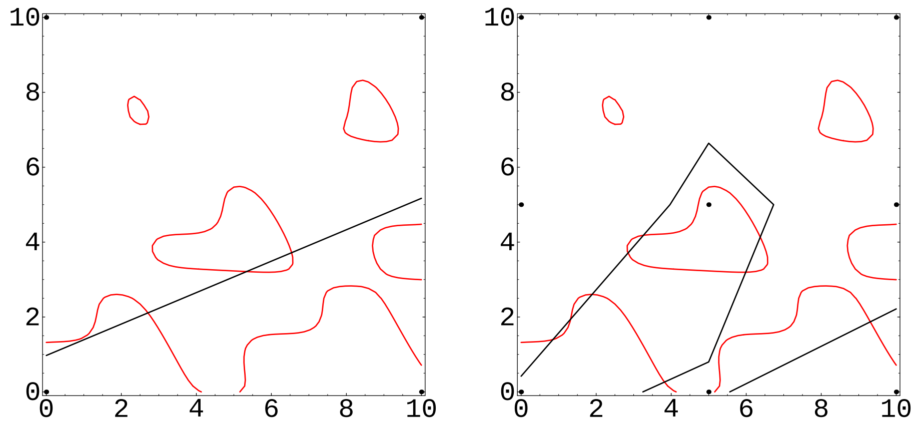
```
      Mod[Take[#, 2], size * 2^-lev] == {0, 0} &];
    posnew = Flatten[Select[am[[2]],
        Module[{thenext}, thenext = Select[amchoice, Function[a,
             Take[#, 2] + {size, 0} * 2^-lev == Take[a, 2]], 1];
          thenext =!= {} && #[[3]] * thenext[[1]][[3]] ≤ 0] &] /.
       {x_?NumberQ, y_, f_} → {{x, y - size * 2^-lev}, {x,
          y + size * 2^-lev}, {x + size * 2^-lev, y - size * 2^-lev},
         {x + size * 2^-lev, y + size * 2^-lev}}, 1];
    posnew = Join[posnew, Flatten[Select[am[[2]],
         Module[{thenext}, thenext = Select[amchoice, Function[a,
              Take[#, 2] + {0, size} * 2^-lev == Take[a, 2]], 1];
           thenext =!= {} && #[[3]] * thenext[[1]][[3]] ≤ 0] &] /.
        {x_?NumberQ, y_, f_} → {{x - size * 2^-lev, y},
          {x + size * 2^-lev, y},
          {x - size * 2^-lev, y + size * 2^-lev},
          {x + size * 2^-lev, y + size * 2^-lev}}, 1]];
    posnew = Union[Select[posnew, #[[1]] ≥ 0 &&
         #[[1]] ≤ size && #[[2]] ≥ 0 && #[[2]] ≤ size &]];
    {numnew, am} = InsertNew[am, posnew];
    MapAt[# + numnew &, am, {1, 4}]];
MyRefine[main_] :=
  Module[{am}, am = main; For[i = 1, i ≤ am[[1]][[2]],
    i++, am = RefDiv[am, i]; am = RefAdd[am, i]]; am];
Step[main_] := Module[{am, ifrefine, orig}, am = main; ifrefine =
    am[[1]][[4]] / Plus @@ am[[1]][[{3, 4}]] < proporref;
   If[ifrefine, am = MapAt[# + 1 &, am, {1, 2}]];
    orig = am[[1]][[4]]; am = FixedPoint[MyRefine[#] &, am]];
   If[! ifrefine || orig == am[[1]][[4]], am = Uni[am];
    am = FixedPoint[MyRefine[#] &, am];
    am = MapAt[Max[#, am[[1]][[1]]] &, am, {1, 2}]];
   am];
interp[main_, x_, y_] :=
 main[[2]][[Position[#, Min[#]][[1, 1]] &[
     Norm[{x, y} - Take[#, 2]] & /@ main[[2]]]]][[3]]
depict[main_] := Module[{p1, p2}, p1 =
    ContourPlot[interp[main, x, y], {x, 0, size}, {y, 0, size},
     PlotPoints → 2^main[[1]][[2]] + 1, Contours → {0},
     ContourShading → None, DisplayFunction → Identity];
   p2 = Graphics[{PointSize[.012],
      main[[2]] /. {x_?NumberQ, y_, f_} → Point[{x, y}]}];
   Show[p, p1, p2, DisplayFunction → Identity,
    PlotRange → {{-.1, size + .1}, {-.1, size + .1}}]];
```

In[41]:=

```
Val = {{{0, 0, 4, 0},
    Flatten[Table[{j * size, i * size, f[j * size, i * size]},
      {i, 0, 1}, {j, 0, 1}], 1]}};
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
  allc = (2^Last[Val][[1]][[2]] + 1)^2},
 ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
  ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
pic1 = depict[Last[Val]];
Val = Append[Val, Step[Last[Val]]];
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
  allc = (2^Last[Val][[1]][[2]] + 1)^2},
 ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
  ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
pic2 = depict[Last[Val]];
Show[GraphicsArray[{pic1, pic2}]];
```

Out[42]= {0, 0, 4, 0}, 4 / 4 (100.%)

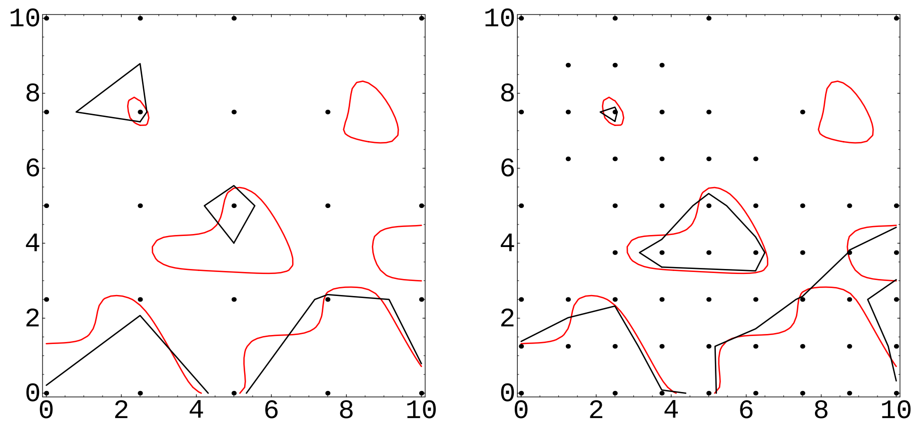
Out[45]= {0, 1, 4, 5}, 9 / 9 (100.%)



In[48]:=
```
Val = Append[Val, Step[Last[Val]]];
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
  allc = (2^Last[Val][[1]][[2]] + 1)^2},
 ToString[Last[Val][[1]]] <> ",  " <> ToString[eval] <> "  /  " <>
  ToString[allc] <> "  (" <> ToString[eval / allc * 100.] <> "%)"]
pic1 = depict[Last[Val]];
Val = Append[Val, Step[Last[Val]]];
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
  allc = (2^Last[Val][[1]][[2]] + 1)^2},
 ToString[Last[Val][[1]]] <> ",  " <> ToString[eval] <> "  /  " <>
  ToString[allc] <> "  (" <> ToString[eval / allc * 100.] <> "%)"]
pic2 = depict[Last[Val]];
Show[GraphicsArray[{pic1, pic2}]];
```

Out[49]= {0, 2, 4, 19}, 23 / 25 (92.%)

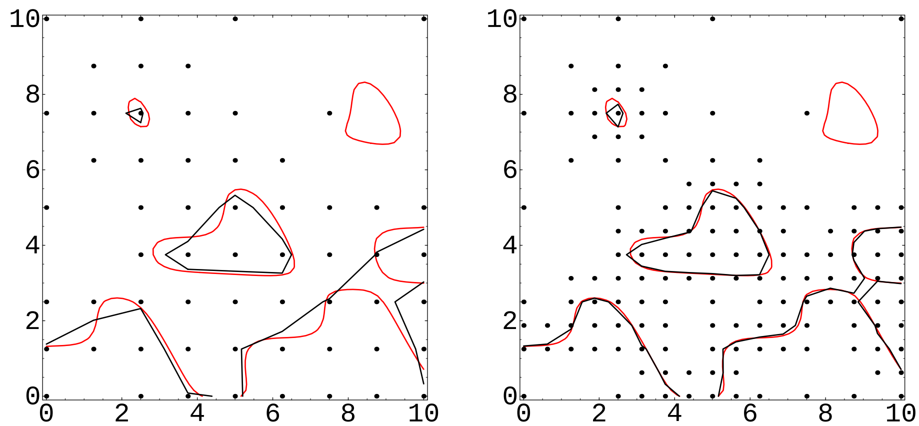
Out[52]= {0, 3, 4, 55}, 59 / 81 (72.8395%)



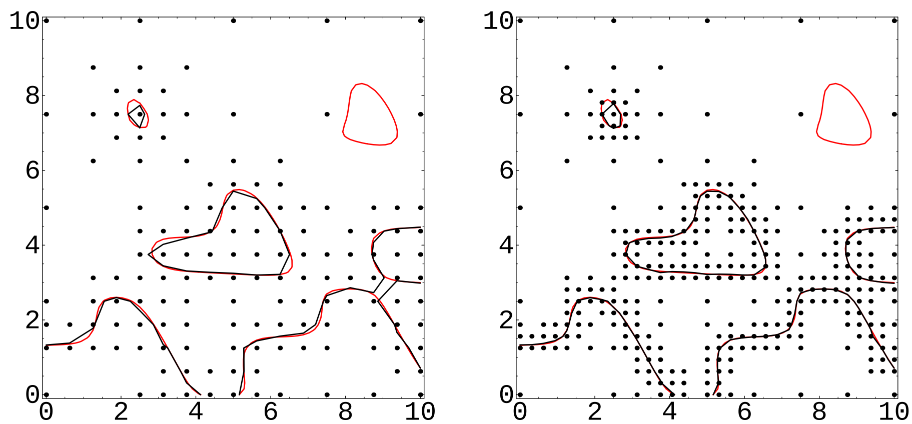
In[55]:=
```
Val = Append[Val, Step[Last[Val]]];
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
   allc = (2^Last[Val][[1]][[2]] + 1)^2},
  ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
   ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
pic1 = depict[Last[Val]];
Val = Append[Val, Step[Last[Val]]];
Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
   allc = (2^Last[Val][[1]][[2]] + 1)^2},
  ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
   ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
pic2 = depict[Last[Val]];
Show[GraphicsArray[{pic1, pic2}]];
```

Out[56]= {1, 3, 9, 50}, 59 / 81 (72.8395%)

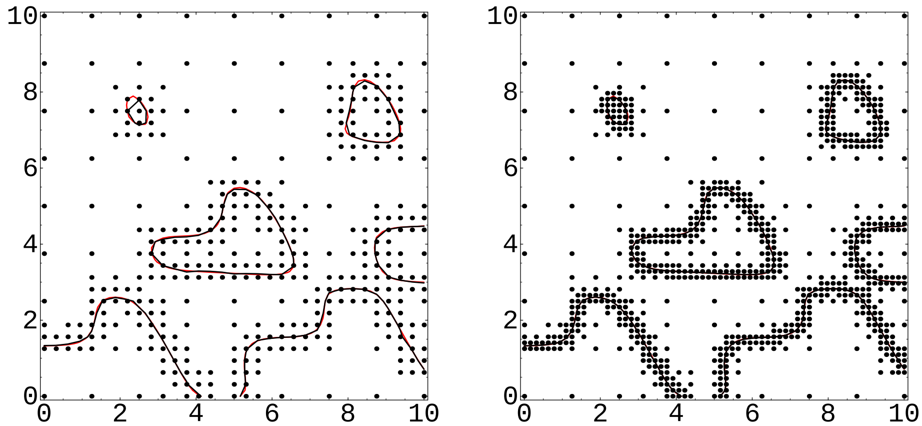
Out[59]= {1, 4, 9, 135}, 144 / 289 (49.827%)



```
In[62]:= Val = Append[Val, Step[Last[Val]]];
         Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
           allc = (2^Last[Val][[1]][[2]] + 1)^2},
          ToString[Last[Val][[1]]] <> ",  " <> ToString[eval] <> "  /  " <>
           ToString[allc] <> "  (" <> ToString[eval / allc * 100.] <> "%)"]
         pic1 = depict[Last[Val]];
         Val = Append[Val, Step[Last[Val]]];
         Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
           allc = (2^Last[Val][[1]][[2]] + 1)^2},
          ToString[Last[Val][[1]]] <> ",  " <> ToString[eval] <> "  /  " <>
           ToString[allc] <> "  (" <> ToString[eval / allc * 100.] <> "%)"]
         pic2 = depict[Last[Val]];
         Show[GraphicsArray[{pic1, pic2}]];

Out[63]= {2, 4, 25, 121}, 146 / 289 (50.519%)

Out[66]= {2, 5, 25, 296}, 321 / 1089 (29.4766%)
```

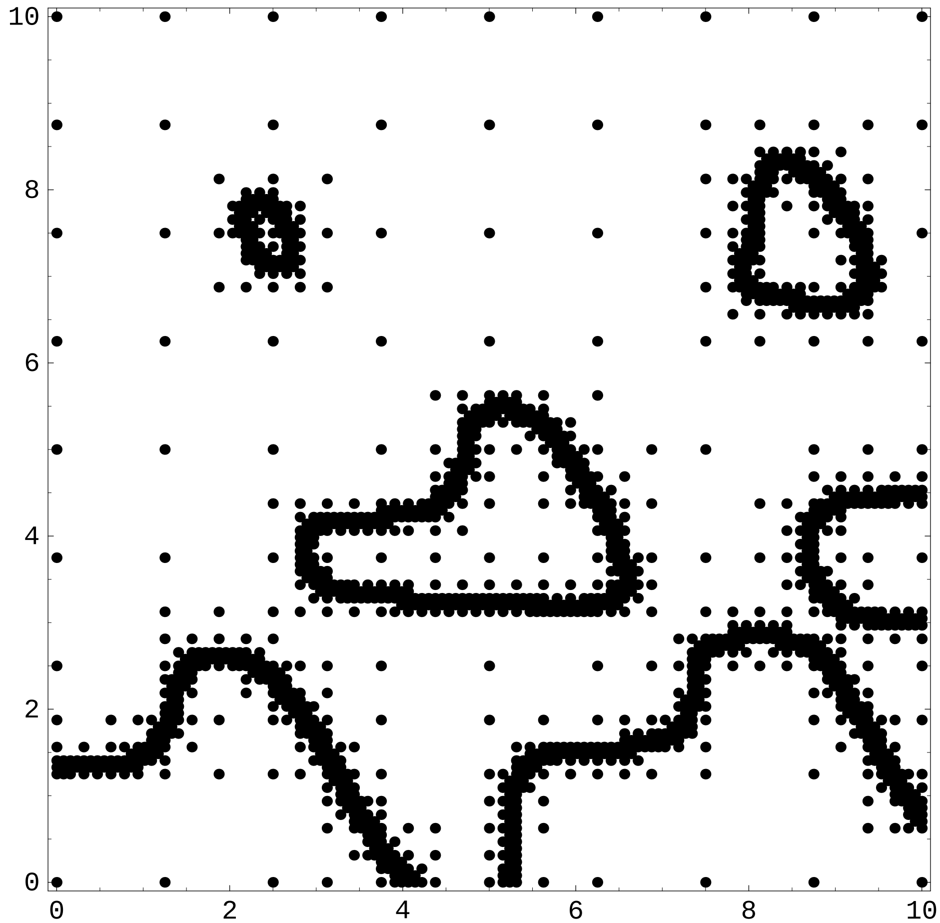


```
In[69]:= Val = Append[Val, Step[Last[Val]]];
         Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
           allc = (2^Last[Val][[1]][[2]] + 1)^2},
          ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
           ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
         pic1 = depict[Last[Val]];
         Val = Append[Val, Step[Last[Val]]];
         Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
           allc = (2^Last[Val][[1]][[2]] + 1)^2},
          ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
           ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
         pic2 = depict[Last[Val]];
         Show[GraphicsArray[{pic1, pic2}]];

Out[70]= {3, 5, 81, 302}, 383 / 1089 (35.1699%)

Out[73]= {3, 6, 81, 719}, 800 / 4225 (18.9349%)
```



We get the same result, needing only 19% of the points, that standard **`ContourPlot`** needed. Of course, we can continue in the refinement.

```
In[76]:= Val = Append[Val, Step[Last[Val]]];
         Block[{eval = Plus @@ Last[Val][[1]][[{3, 4}]],
           allc = (2^Last[Val][[1]][[2]] + 1)^2},
          ToString[Last[Val][[1]]] <> ", " <> ToString[eval] <> " / " <>
           ToString[allc] <> " (" <> ToString[eval / allc * 100.] <> "%)"]
         Show[depict[Last[Val]], DisplayFunction → $DisplayFunction];

Out[77]= {3, 7, 81, 1548}, 1629 / 16641 (9.78908%)
```



The author is a mathematician working in the theory of existence, uniqueness and bifurcations of nontrivial solutions of higher−order nonlinear differential equations.