In [68]:
```
%load_ext sympyprinting
# import the AnsatzFunction class
from AnsatzFunction import AnsatzFunction
from Plotting import plotHierarchical1d
```

Exercise 1: One-dimensional Sparse Grids---An Adaptiave Implementation

We want to use an adaptiave version of Archimedes' approach to approximate the integral $F(f, a, b) = \int_a^b f(x)\, dx$ of a function $f : \mathbb{R} \to \mathbb{R}$ or to approximate the function f itself.

For the one-dimensional case we want to formalize this approach and generalize it in the following ways:

- Let $\phi(x)$ be the mother of all hat functions with

In [69]:
```
AnsatzFunction(0,0).getPhi()
```

Out[69]:

$$\begin{cases} 0 & \text{for}\, x < -1.0 \\ x + 1 & \text{for}\, x < 0 \\ -x + 1 & \text{for}\, x < 1.0 \\ 0 & \text{otherwise} \end{cases}$$

- The data structure used to store the hierarchical coefficients is now called *Sparse Grid*.
- A sparse grid is defined by a particular set of interpolation points $x_{l,i}$ and associated ansatz functions $\phi_{l,i}(x)$ with
  $\phi_{l,i}(x) = \phi(\, 2^l \cdot \left(x - i \cdot \frac{1}{2^l}\right)\,) = \phi(2^l \cdot x - i),\ l \in \mathbb{N}^+, 1 \leq i \leq 2^l - 1, i \text{ odd}$
- Archimedes' approach from the lecture corresponds to a *regular* sparse grid.
- To improve the quality of approximation for arbitrary functions f we introduce spatial adaptivity.

Import and use the class `AnsatzFunction` and look at the comments in the provided code snippets for some more details. Use it to construct a one-dimensional sparse grid structure, which stores its grid points (resp. ansatz functions of type `AnsatzFunction` in a hash map.

In [70]:
```
# the function we want to approximate
def parabola(x):
    return 4*x*(1-x)

def asymmetric(x):
    return 8*(-16*x**4 + 40*x**3 -35*x**2 + 11*x)/9

func = asymmetric

def printSparseGrid(sg):
    for af in sg.itervalues():
        x = af.computeCoordinate()
        print af.getKey(), ": u(%f)=%f\tu_h(%f)=%f" % (x, af.getFunctionValue(),

def evaluateSparseGrid(sg, x):
    '''Evaluates a given sparse grid at given point x'''
    return sum(map(lambda ansatz: ansatz(x), sg.itervalues()))


def applyRefinementCriterion(sg, criterion, func):
    '''
    Applies the criterion to the leaves of the sparse grid.

    Calls the "apply" member for each leaf of the grid,
    then finishes calling the "finalize" member passing the
    grid and the approximated function as arguments.
    We'll also use it to build our sparse grid.
    '''
    # iterate over leaves only
    for key in sg:
        leftChild =  (key[0]+1, 2*key[1]-1)
        rightChild = (key[0]+1, 2*key[1]+1)
        if not (sg.has_key(leftChild) and sg.has_key(rightChild)):
            criterion.apply(sg[key])
    criterion.finalize(sg, func)
```

We start by creating an empty sparse grid.

In [71]:
```
sg = {}
```

We implement a refinement criterion that fills the sparse grid with points up to a given level.

In [72]:
```
class MinLevelCriterion(object):
    '''Criterion adding all missing points up to a certain level.'''
    def __init__(self, minLevel):
        self.minLevel = minLevel
    def apply(self, af):
        pass
    def finalize(self, sg, func):
        for l in xrange(1, self.minLevel+1):
            for i in xrange(1, 2**l, 2):
                if not sg.has_key((l,i)):
                    x = float(i)/2**l
                    f_x = func(x)
                    sg[(l,i)] = AnsatzFunction(l, i, f_x, f_x)
```

In [73]:
```
# fill the sparse grid
applyRefinementCriterion(sg, MinLevelCriterion(3), func)
printSparseGrid(sg)
```

```
(3, 3) : u(0.375000)=0.885417   u_h(0.375000)=0.885417
(3, 1) : u(0.125000)=0.802083   u_h(0.125000)=0.802083
(2, 1) : u(0.250000)=1.000000   u_h(0.250000)=1.000000
(2, 3) : u(0.750000)=0.333333   u_h(0.750000)=0.333333
(3, 7) : u(0.875000)=0.218750   u_h(0.875000)=0.218750
(1, 1) : u(0.500000)=0.666667   u_h(0.500000)=0.666667
(3, 5) : u(0.625000)=0.468750   u_h(0.625000)=0.468750
```

We implement the hierarchization method.

In [74]:
```
def hierarchize(sg):
    '''Recursive hierarchization function.'''
    def rec(l, i, f_l, f_r):
        if sg.has_key((l,i)):
            af = sg[(l,i)]
            f_m = af.getSurplus()
            rec(l+1, 2*i-1, f_l, f_m)
            rec(l+1, 2*i+1, f_m, f_r)
            af.setSurplus(f_m - 0.5*(f_l+f_r))
    rec(1, 1, 0.0, 0.0)
```

In [75]:
```
# apply hierarchization
hierarchize(sg)
printSparseGrid(sg)
```

```
(3, 3) : u(0.375000)=0.885417   u_h(0.375000)=0.052083
(3, 1) : u(0.125000)=0.802083   u_h(0.125000)=0.302083
(2, 1) : u(0.250000)=1.000000   u_h(0.250000)=0.666667
(2, 3) : u(0.750000)=0.333333   u_h(0.750000)=0.000000
(3, 7) : u(0.875000)=0.218750   u_h(0.875000)=0.052083
(1, 1) : u(0.500000)=0.666667   u_h(0.500000)=0.666667
(3, 5) : u(0.625000)=0.468750   u_h(0.625000)=-0.031250
```

Define a more complex refinement criterion looking for large surpluses.

In [76]:

```
class EpsilonCriterion(object):
    '''Criterion refining points with large surpluses.'''
    def __init__(self, eps):
        self.eps = eps
        self.toRefine = []
    def apply(self, af):
        if abs(af.getSurplus()) > self.eps:
            self.toRefine.append(af.getKey())
    def finalize(self, sg, func):
        for (l,i) in self.toRefine:
            child = [l+1,2*i-1]
            if not sg.has_key(tuple(child)):
                af = AnsatzFunction(l+1, 2*i-1)
                x = af.computeCoordinate()
                f_x = func(x)
                af.setFunctionValue(f_x)
                af.setSurplus(f_x - evaluateSparseGrid(sg, x))
                sg[(l+1,2*i-1)] = af
            child[1] = 2*i+1
            if not sg.has_key(tuple(child)):
                af = AnsatzFunction(l+1, 2*i+1)
                x = af.computeCoordinate()
                f_x = func(x)
                af.setFunctionValue(f_x)
                af.setSurplus(f_x - evaluateSparseGrid(sg, x))
                sg[(l+1,2*i+1)] = af
```

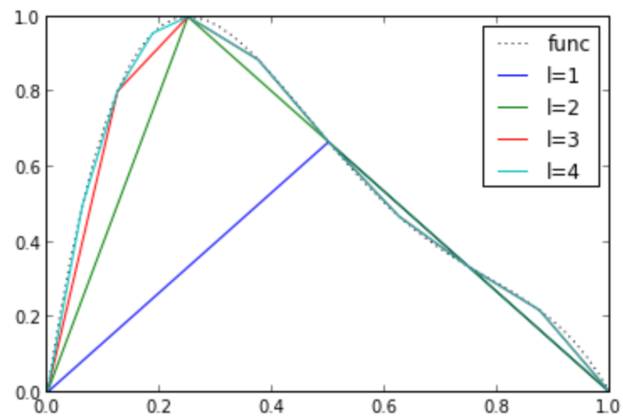
In [77]:

```
# apply the criterion
applyRefinementCriterion(sg, EpsilonCriterion(0.06), func)
printSparseGrid(sg)
```

```
(3, 3) : u(0.375000)=0.885417   u_h(0.375000)=0.052083
(3, 1) : u(0.125000)=0.802083   u_h(0.125000)=0.302083
(2, 1) : u(0.250000)=1.000000   u_h(0.250000)=0.666667
(2, 3) : u(0.750000)=0.333333   u_h(0.750000)=0.000000
(4, 3) : u(0.187500)=0.956380   u_h(0.187500)=0.055339
(3, 7) : u(0.875000)=0.218750   u_h(0.875000)=0.052083
(4, 1) : u(0.062500)=0.498047   u_h(0.062500)=0.097005
(1, 1) : u(0.500000)=0.666667   u_h(0.500000)=0.666667
(3, 5) : u(0.625000)=0.468750   u_h(0.625000)=-0.031250
```

Finally, have a look at the sparse grid.

In [78]:
```
xvec = linspace(0,1,50)
plot(xvec, map(func, xvec), "k:", label="func")
plotHierarchical1d(sg)
```



In [78]: