In [30]:
```
%load_ext sympyprinting
```

Exercise 1: Analytical Integration

Consider the functions

$f(x) = -4\,x\,(x-1)$

and

$g(x) = \frac{8}{9} \cdot (-16\,x^4 + 40\,x^3 - 35\,x^2 + 11\,x)$

Compute the antiderivatives and evaluate the integrals.

In [31]:
```
from __future__ import division
from sympy import *
x, y = symbols('x y')
l, i = symbols('l, i', integer=True)
g, h = symbols('f g', cls=Function)
```

In [32]:
```
f = -4*x*(x-1)
g = 8 * (-16*x**4 + 40*x**3 -35*x**2 + 11*x) / 9
```

In [33]:
```
integrate(g, x)
```

Out[33]:
$$-\frac{128}{45}\,x^5 + \frac{80}{9}\,x^4 - \frac{280}{27}\,x^3 + \frac{44}{9}\,x^2$$

In [34]:
```
integrate(g, (x,0,1))
```

Out[34]:
$$\frac{76}{135}$$

Exercise 2: Composite Trapezoidal Rule

Write a function that approximates the integral via the Composite Trapezoidal Rule. Complete the function template, with a, b and f according to the definition above and n being the number of trapezoids used.

In [35]:
```
def trapezoidalRule(f, a, b, n):
    '''
    Implements CTR to approximate the integral of a given function.
    '''
    result = 0.5*(f(a) + f(b))
    h = (b-a) / float(n)
    for i in xrange(1, n):
        result += f(a + i*h)
    return result * h
```

In [36]:
```
trapezoidalRule(lambda t: f.subs(x, t), 0.0, 1.0, 1000)
```

Out[36]:
$$0.666666$$

Exercise 3: Composite Simpson Rule

Do the same as in Exercise 2 for the Composite Simpson Rule.

In [37]:
```
def simpsonRule(f, a, b, n):
    '''
    Implements SR to approximate the integral of given function.
    '''
    # the boundary function values are weighted with 0.5 - for now
    result = 0.5 * (f(a) + f(b))
    # loop over the center points of the subintervals
    h_2 = (b - a) / (2 * n)         # half width of subinterval
    for i in xrange(1, 2 * n, 2):
        result += 2.0 * f(a + i * h_2)
    # loop over the boundaries of the subintervals
    h = 2.0 * h_2                   # full width of subinterval
    for i in xrange(1, n):
        result += f(a + i * h)
    return h * result / 3.0
```

In [38]:
```
simpsonRule(lambda t: g.subs(x, t), 0.0, 1.0, 5)
```

Out[38]:
$$0.562773333333335$$

Exercise 4: Archimedes' Hierarchical Approach

In this exercise we will use Archimedes' approach to approximate the integral.

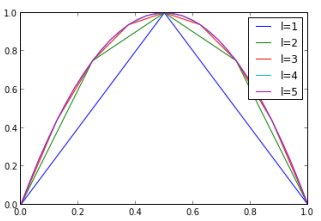
Let $\vec{u} = [u_0, \ldots, u_n]^T \in \mathbb{R}^n, n = 2^l - 1, l \in \mathbb{N}$ a vector of function values with $u_i = f(x_i = \frac{i+1}{2^l})$.

a) Write a function that transforms a given vector $\vec{u} \in \mathbb{R}^n$ to a similar vector $\vec{v} \in \mathbb{R}^n$ containing the hierarchical coefficients needed for Archimedes' quadrature approach. Use the function template below.

In [39]:
```
def hierarchize1d(u, l):
    '''
    Transforms the (2^l-1) func. values in u to hier. coeffs
    '''
    # we copy u to vector v and return v
    v = u[:]
    # compute the length of the array
    n = (1 << l) - 1
    # Hierarchization is done bottom up, i.e. highest refinement level first!
    # Actually start on second highest level and process the level below.
    # Proceed with levels above, always computing the surpluses on the level
    # below, respectively.
    for level in xrange(l - 1, 0, -1):
        # index of the first coefficient on this level
        first = (1 << (l - level)) - 1
        # distance in array to direct children
        delta = 1 << (l - level - 1)
        for j in xrange(first, n, delta << 1):
            v[j - delta] -= 0.5 * v[j]
            v[j + delta] -= 0.5 * v[j]
    return v
```

In [40]:
```
level = 5
xvec = [ i / float(2**level) for i in xrange(1, 1 << level) ]
fvec = map(lambda t: f.subs(x,t), xvec)
v = hierarchize1d(fvec, level)
```

In [43]:
```
plotHierarchical1d(v, level, basis="linear", flat=False)
```



In [43]:
```
```