Topological_conjugacy_1

February 4, 2019

1 Topological Conjugacy for homeomorphisms of $\mathbb{R}$.

Let $f : I \to I$ and $g : J \to J$ be continuous maps. We say they are topologically conjugate if there is a homeomorphism $h : I \to J$ so that $g \circ h(x) = h \circ f(x)$ for all $x \in I$.

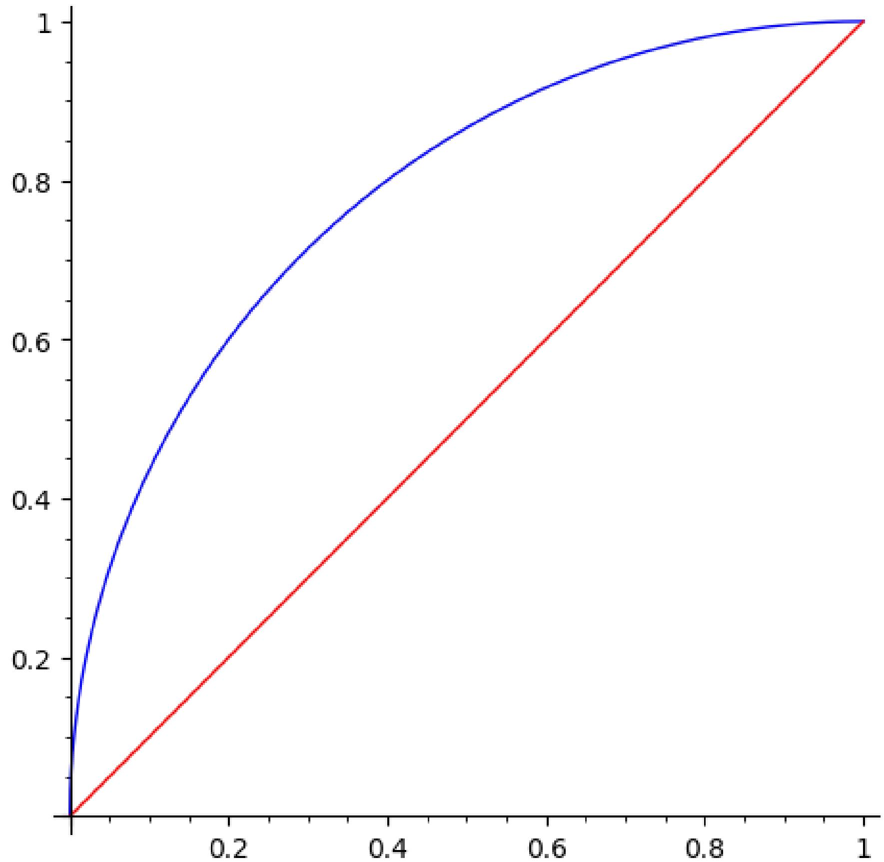
We will demonstrate the idea behind the following result:

Theorem. Suppose $I = (a, b)$ and $J = (c, d)$ are intervals in $\mathbb{R}$. Suppose $f : I \to I$ and $g : J \to J$ are orientation-preserving homeomorphisms so that * $f(x) > x$ for each $x \in I$, and * $g(y) > y$ for each $y \in J$. Then f and g are topologically conjugate.

To demonstrate this, we will consider two such maps.

```
In [1]: f(x) = sqrt(2*x-x^2) # Consider over the interval (0,pi)
        plot(f, 0, 1, aspect_ratio=1) + plot(x,(x, 0, 1), color="red")
```

```
Out[1]:
```



In [2]:
```
# Here we work out the inverse map
x=var("x")
y=var("y")
assume(y>0) # Used to help Sage find the solution we want below.
assume(y<1)
show((f(x)==y).solve(x))
```

```
[x == -sqrt(-y^2 + 1) + 1, x == sqrt(-y^2 + 1) + 1]
```

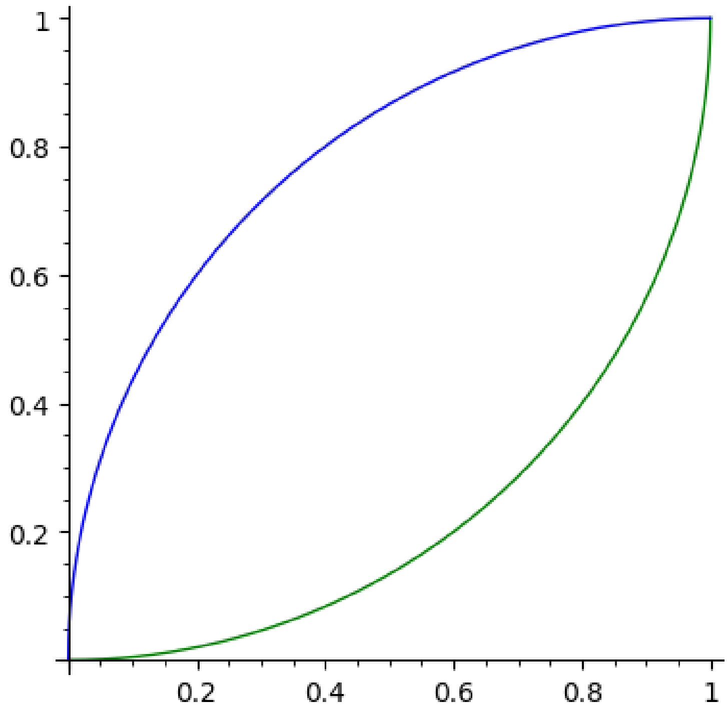
Note that the inverse must be the first one since the second takes values greater than one.

In [3]:
```
# Here we define the inverse map
finv(y) = -sqrt(-y^2 + 1) + 1
```

Lets plot f^{-1} with f to be sure.

In [4]:
```
plot(finv, 0, 1, color="green", aspect_ratio=1) + plot(f,(x,0,1), color="blue")
```

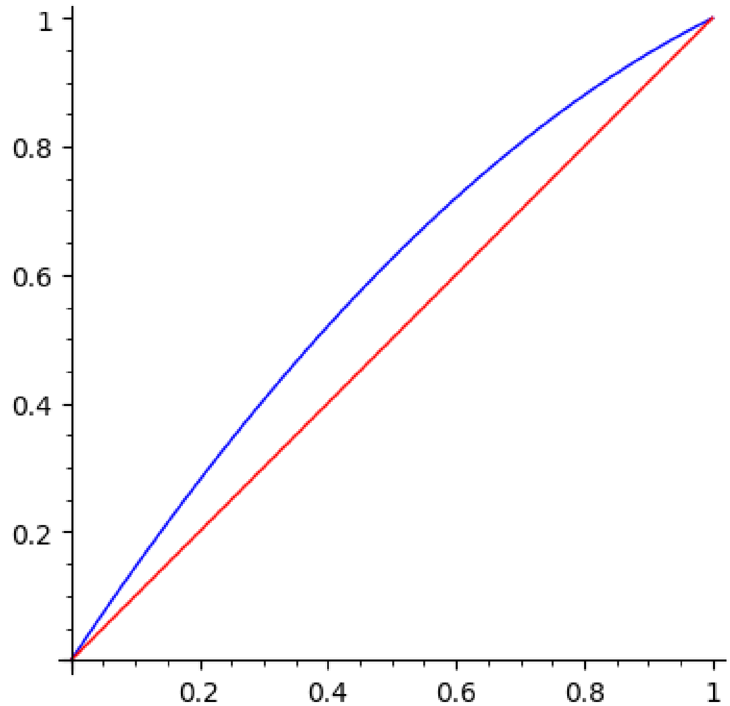
Out[4]:



In [5]:
```
g(x) = 1/2*(x*(3-x)) # Consider over the interval (0,pi)

plot(g, 0, 1, aspect_ratio=1) + plot(x,(x,0,1), color="red")
```

Out[5]:



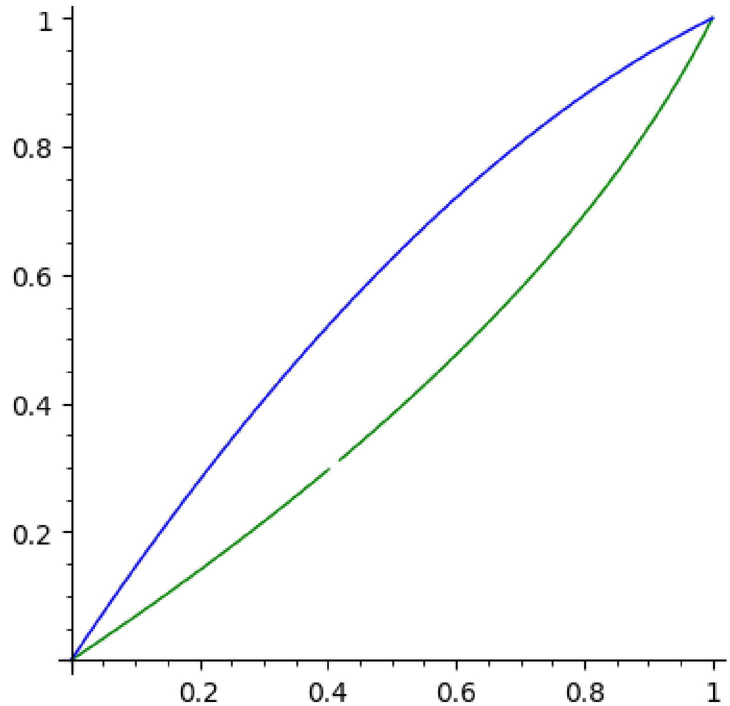

```
In [6]: # Here we work out the inverse map
        x=var("x")
        y=var("y")
        assume(y>0) # Used to help Sage find the solution we want below.
        assume(y<1)
        (g(x)==y).solve(x)
```

```
Out[6]: [x == -1/2*sqrt(-8*y + 9) + 3/2, x == 1/2*sqrt(-8*y + 9) + 3/2]
```

```
In [7]: ginv(y) = -1/2*sqrt(-8*y + 9) + 3/2
```

```
In [8]: plot(ginv, 0, 1, color="green", aspect_ratio=1) + plot(g,(x,0,1), color="blue")
```

```
Out[8]:
```



1.0.1 Defining the topological conjugacy:

First we pick a points a_f and a_g in the domains of f and g:

```
In [9]: a_f = 1/2
        a_g = 1/2
```

We define $b_f = f(a_f)$ and $b_g = g(a_g)$:

```
In [10]: b_f = f(a_f)
         print("b_f = %s"%b_f)
         b_g = g(a_g)
         print("b_g = %s"%b_g)
```

```
b_f = 1/2*sqrt(3)
b_g = 5/8
```

The intervals $[a_f, b_f)$ is a fundamental domains for f. This means for each $x \in (0,1)$, there is a unique $n \in \mathbb{Z}$ so that $f^n(x) \in [a_f, b_f)$. Similarly, $[a_g, b_g)$ is a fundamental domain for g.

We define a homeomorphism $h_0 : [a_f, b_f) \to [a_g, b_g)$.

```
In [11]: h_0(x) = (b_g - a_g)/(b_f - a_f)*(x - a_f) + a_g
         show(h_0(x))
         assert h_0(a_f)==a_g # Prints errors if false.
         assert h_0(b_f)==b_g
```

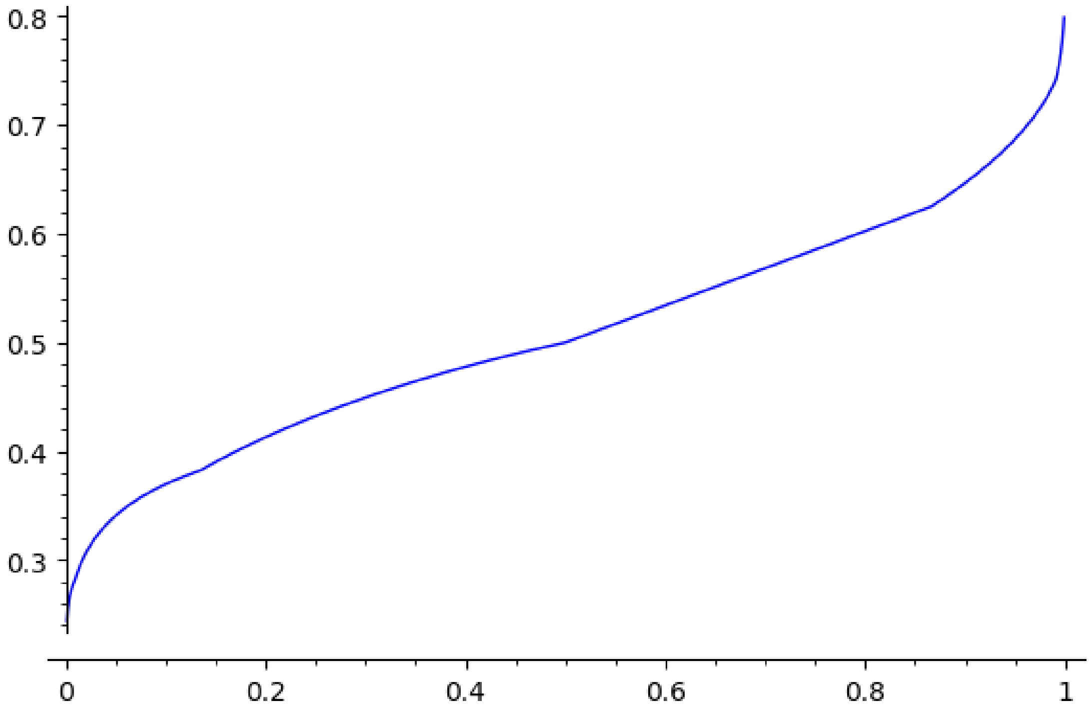
```
1/8*(2*x - 1)/(sqrt(3) - 1) + 1/2
```

Note that this function is more complex, so we define it using a Python type function. This allows us to use any Python or Sage type expression we want, including if statements and loops.

```
In [12]: def h(x):
             assert 0 < x < 1 # Cause an error if not in the domain of f.
             if a_f <= x < b_f:
                 # Use h_0:
                 return h_0(x)
             if x >= b_f:
                 count = 0
                 while x >= b_f:         # Apply f^-1 until we land in the fundamental
                     x = finv(x)         # domain and count the number of times
                     count = count + 1   # we apply f^-1.
                 assert a_f <= x < b_f
                 y = h_0(x)  # Move to the domain of g using h_0.
                 for i in range(count): # Now apply g to y, the same number of times.
                     y = g(y)
                 return y
             if x < a_f:
                 count = 0
                 while x < a_f:
                     x = f(x)
                     count = count + 1
                 assert a_f <= x < b_f
                 y = h_0(x)
                 for i in range(count):
                     y = ginv(y)
                 return y
```

```
In [13]: # Plot h.
         # Note that h is not defined at zero or at one, so
         # we have shrunk the interval we are plotting slightly.
         # Calling plot(h, 0, 1) will give rise to errors.
         plot(h, 0.001, 0.999)
```

Out[13]:




```
In [14]: # Check one value larger than b_f:
         show( h(9/10) )
         # Check the conjugacy equation:
         assert( h(f(9/10)) == g(h(9/10)) )
```

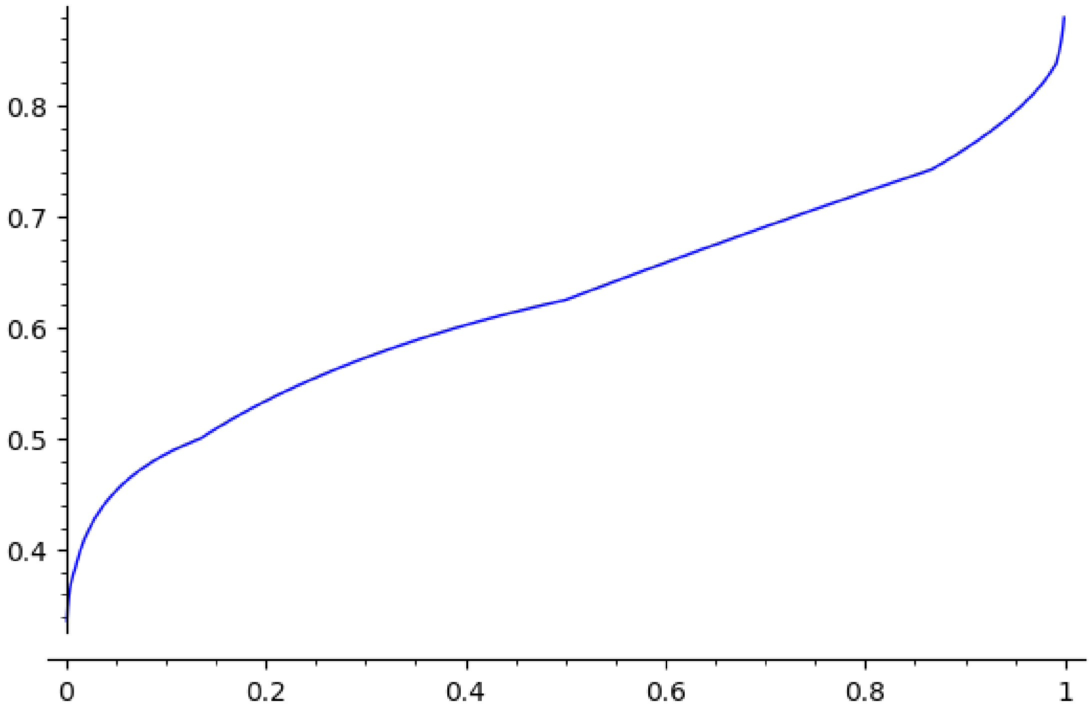
```
-1/3200*((sqrt(19) - 5)/(sqrt(3) - 1) + 100)*((sqrt(19) - 5)/(sqrt(3) - 1) - 20)
```

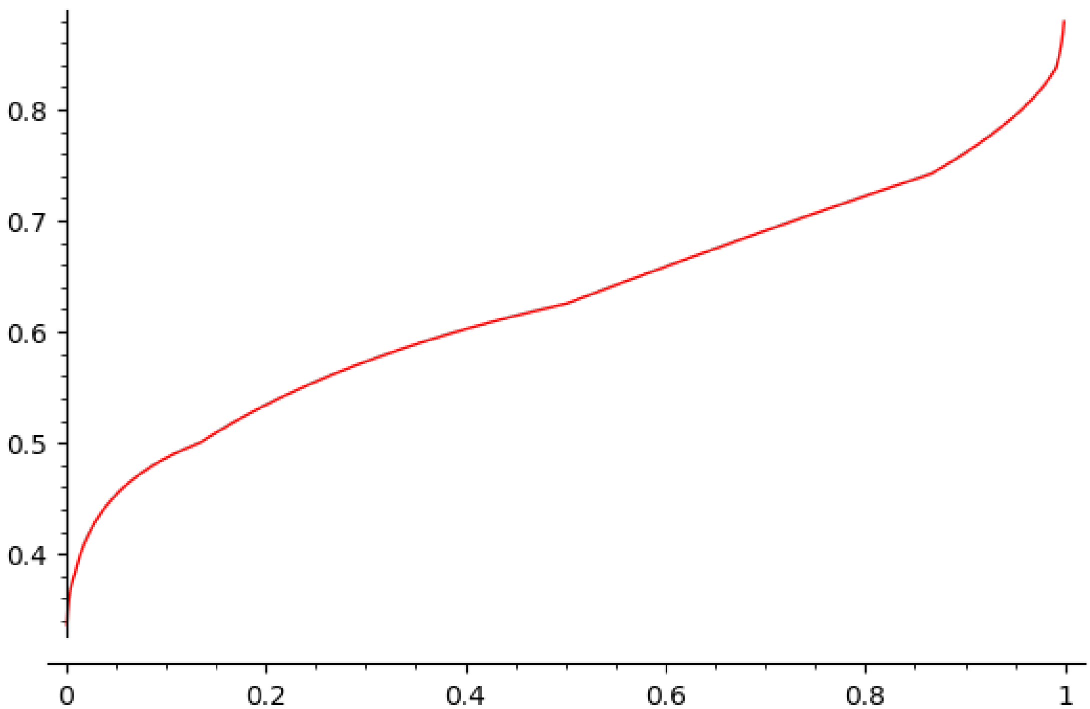
```
In [15]: # Check one value larger than b_f:
         show( h(1/4) )
         # Check the conjugacy equation:
         assert( h(f(1/4)) == g(h(1/4)) )
```

```
-1/2*sqrt(-1/2*(sqrt(7) - 2)/(sqrt(3) - 1) + 5) + 3/2
```

We can graphically check the conjugacy. For plot1 we will plot $h \circ f$ and for plot2 we will plot $g \circ h$.

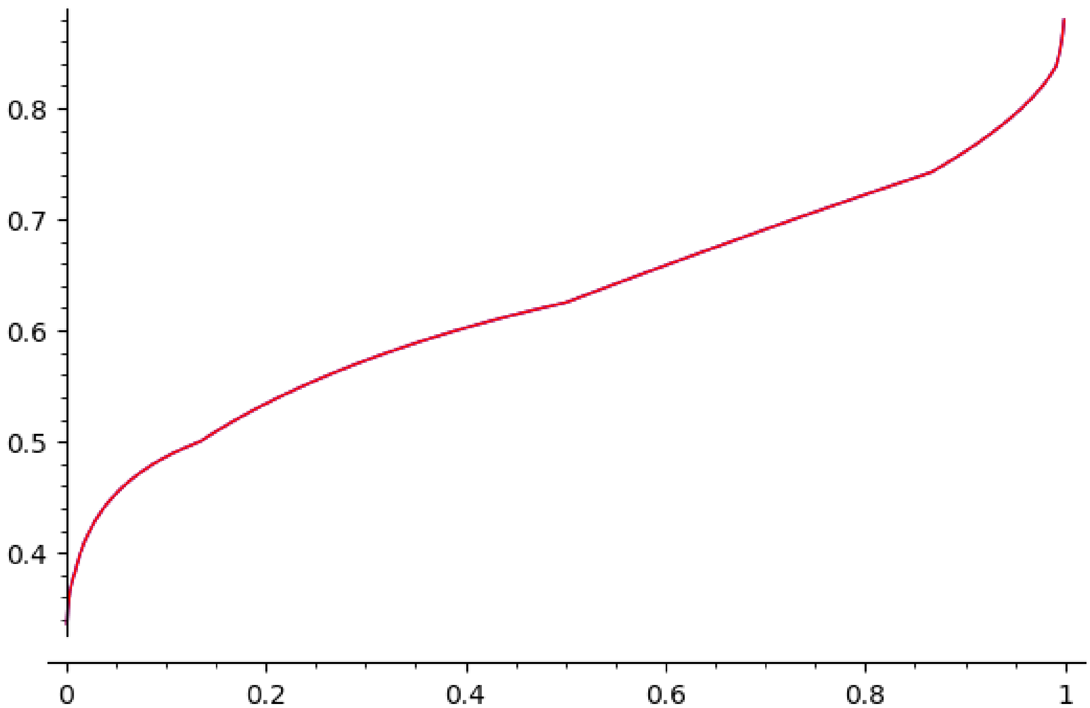
```
In [16]: plot1 = plot(lambda x: h(f(x)), 0.001, 0.999)
         show(plot1)
         plot2 = plot(lambda x: g(h(x)), 0.001, 0.999, color="red")
         show(plot2)
```


0.8
0.7
0.6
0.5
0.4
0
0.2
0.4
0.6
0.8
1


0.8
0.7
0.6
0.5
0.4
0
0.2
0.4
0.6
0.8
1

In [17]:
```
# Plot them on top of each other
plot1 + plot2
```

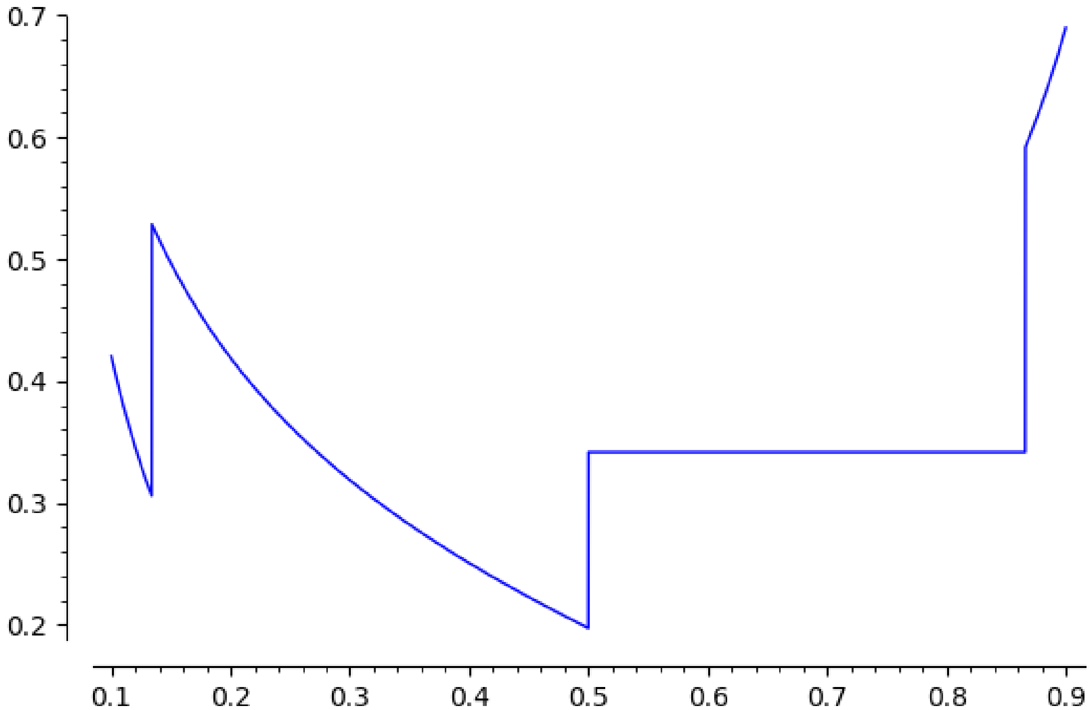
Out[17]:



In [18]:
```
# Note that h is continuous but not differentiable:
def approximate_derivative_of_h(x, epsilon=0.0001):
    return (h(x+epsilon)-h(x)) / epsilon
```

In [19]:
```
plot(approximate_derivative_of_h,0.1,0.9)
```

Out[19]:



By fiddling appropriately with our function $h_0(x)$ we could get h to be smooth. The main issue is derivatives at a_f. At other points, h is defined to be h_0 or compositions of h_0 with powers of f and g. Note that for values slightly bigger than a_f, h is given by h_0. While for values slightly to the left of a_f, h is given by $g^{-1} \circ h_0 \circ f$. Thus if we want the derivative to match at a_f, we would have to have

$$h_0'(a_f) = (g^{-1})'\big(h \circ f(a_f)\big) \cdot h_0'\big(f(a_f)\big) \cdot f'(a_f) = (g^{-1})'(b_g)h_0'(b_f)f'(a_f) = \frac{f'(a_f)}{g'(a_g)}h_0'(b_f).$$

So we would have to choose h_0 to satisfy $g'(a_g)h_0'(a_f) = f'(a_f)h_0'(b_f)$.