Aitkin's δ^2-method

In [2]:
```
m=10
F(x)=(((-3*x-2)*x+m)*x-1)/(m+1)
```

simple scheme that happened to converge linearly...

Out[2]: F (generic function with 1 method)

In [4]:
```
x0=-0.75
x1=F(x0)
x2=F(x1);
```

In [6]:
```
println("x0=",x0," x1=",x1," x2=",x2)
```

```
x0=-0.75 x1=-0.7599431818181818 x2=-0.7670751308110003
```

In [7]:
```
d2(xn,xnp1,xnp2)=xnp2-(xnp1-xnp2)^2/(xnp2-2*xnp1+xn)
```

Out[7]: d2 (generic function with 1 method)

In [8]:
```
x3=d2(x0,x1,x2)
```

Out[8]: -0.7851685081994753

In [9]:
```
x4=d2(x1,x2,x3)
```

Out[9]: -0.7553028471281954

That was the wrong thing to do because x_1 and x_2 are bad approximations before the δ^2 leap

In [10]:
```
x4=F(x3)
```

Out[10]: -0.7847745116562741

In [11]:
```
x5=F(x4)
```

Out[11]: -0.7845025013326782

Note these numbers x_4 and x_5 are pretty close to x_3 but still converging linearly. So it's time for another leap.

In [12]:
```
x6=d2(x3,x4,x5)
```

Out[12]:
```
-0.7838959605579727
```

In [13]:
```
x7=F(x6)
x8=F(x7)
x9=d2(x6,x7,x8)
```

Out[13]:
```
-0.7838942936898103
```

Note from last time evaluating $x_6 = F^6(x_0) = -0.779942560069474$. Even if F is quick to evaluate, then compared to $x_9 = F^9(x_0)$ this is still a significant improvement in efficiency.

In [23]:
```
x=big(x0);
for n=3:3:18
    t1=x; t2=F(t1); t3=F(t2)
    x=d2(t1,t2,t3)
    println(n,"  ",x)
end
```

using the delta^2 method to accelerate the linearly convergent scheme...

```
3  -0.7851685081994759238318414873953777753557042103207582301302924199693910452911658
6  -0.7838959605579724906674765949119839150898085539296146126717755966034257479871 38
9  -0.7838942936898097717955265099916553102598515236028409777082302773003450703411728
12  -0.7838942936869493834290851056943440302970535469385724877383507570414630442924649
15  -0.7838942936869493834290766825820014003310776218152335543801156401586855427067863
18  -0.7838942936869493834290766825820014003310776217421923756433580667834252892281839
```

In [27]:
```
xs=big.(zeros(6))
```

Out[27]:
```
6-element Array{BigFloat,1}:
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
```

In [30]:
```
xs=big.(zeros(6))
x=big(x0)
for n=3:3:18
    t1=x; t2=F(t1); t3=F(t2)
    x=d2(t1,t2,t3)
    xs[n ÷ 3]=x
    println(n,"  ",x)
end
```

```
3  -0.7851685081994759238318414873953777753557042103207582301302924199693910452911658
6  -0.7838959605579724906674765949119839150898085539296146126717755966034425747987138
9  -0.7838942936898097717955265099916553102598515236028409777082302773003450703411728
12  -0.7838942936869493834290851056943440302970535469385724877383507570414630442924649
15  -0.7838942936869493834290766825820014003310776218152335543801156401586855427067863
18  -0.7838942936869493834290766825820014003310776217421923756433580667834252892281839
```

In [31]:
```
xs
```

Out[31]:
```
6-element Array{BigFloat,1}:
 -0.7851685081994759238318414873953777753557042103207582301302924199693910452911658
 -0.7838959605579724906674765949119839150898085539296146126717755966034425747987138
 -0.7838942936898097717955265099916553102598515236028409777082302773003450703411728
 -0.7838942936869493834290851056943440302970535469385724877383507570414630442924649
 -0.7838942936869493834290766825820014003310776218152335543801156401586855427067863
 -0.7838942936869493834290766825820014003310776217421923756433580667834252892281839
```

In [35]:
```
es=xs[1:5].-xs[6]
```

Out[35]:
```
5-element Array{BigFloat,1}:
 -0.0012742145125265404027648048133763750246265885785658544869343531859657560629819 29
 -1.6668710231072383999123299825147587309321874222370284175298200004587589541293 98e-06
 -2.8603883664498274096539099287739018606486020648722105169197811129888916938035 03e-12
 -8.4231123426299659759251963801120949926902580377550642809726553615700
```

```
37193463799e-24
 -7.30411787367575733752602534786024407815513363700063736565914430296801
15960680068e-47
```

In [38]:
```
lek=log.(abs.(es))
```

Out[38]:
```
5-element Array{BigFloat,1}:
   -6.665425358826623634010547778548572189790549059336045664068350807488
8334141931558
  -13.304562327850405836548839112180089232907802328012200061308703058136
780889271303
  -26.580063707850774572275908080607619932587172267543263840422902681400
44028996885
  -53.131062835009827034999870247799812095062255736764445334442048828732
08483204278
 -106.233061089321697908666577862689116196799945261400553806016561665224
2779872072
```

In [39]:
```
slope=(lek[5]-lek[4])/(lek[4]-lek[3])
```

Out[39]:
```
1.999999999999765204625582624512112095748424507637037641964926018553253
267489195
```

Numerically, this verifies that the new method is converging quadratically, that is, has order 2 convergence--which is just as fast as Newton's method! Is this really the case in general?

Note, if one where to prove the rate of convergence was really order 2, then Calculus would be needed and likely lots of it! What is the order?

Over the weekend see if you can use Taylor's theorem to explain why this method is order 2 or if not find the general order of the method. Try looking it up or searching online.

In [ ]:
```
```