newton

June 16, 2020

1 Newton method

[1]:
```
using Plots
plotly()
```

[1]:
```
Plots.PlotlyBackend()
```

1.1 Newton-Raphson in optimization

Newton-Raphson iteration

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

[2]:
```
function Newton(f::Function, df::Function, d2f:: Function,
        xstart::Float64, verbose::Bool = false, store=false,
         ::Float64 = 1e-6, nmax::Int64 = 1000)
    k = 1
    x = xstart
    if (store)
        iter = [ f(x) x ]
    end
    if (verbose)
        fx = f(x)
        println("$k. x = $x, f(x) = $fx")
    end
    dfx = df(x)
    while (abs(dfx) >   && k < nmax)
        k += 1
        dfx = df(x)
        x = x-dfx/d2f(x)
        if (store)
            iter = [iter ; f(x) x]
        end
        if (verbose)
            fx = f(x)
            println("$k. x = $x, f(x) = $fx")
        end
    end

    if (store)
        return iter
    end
end
```

[2]:
```
Newton (generic function with 5 methods)
```

[3]:
```
func(x) = -10x^2 + 4sin(x) + x^4
dfunc(x) = -20x + 4cos(x) + 4x^3
d2func(x) = -20 - 4sin(x) + 12x^2
```

[3]:
```
d2func (generic function with 1 method)
```

[4]:
```
plot(func, -4.0, 4.0)
```

If we start close enough to the global optimum, we are to find it.

[5]:
```
x0 = -3.0
iter = Newton(func, dfunc, d2func, x0, true, true)
```

```
1. x = -3.0, f(x) = -9.564480032239473
2. x = -2.413309150943051, f(x) = -26.983281625936534
3. x = -2.2051283480850294, f(x) = -28.202989762523757
4. x = -2.1772606979392806, f(x) = -28.219314490011172
5. x = -2.1767739238815578, f(x) = -28.21931925035788
6. x = -2.1767737764195543, f(x) = -28.21931925035831
7. x = -2.176773776419541, f(x) = -28.219319250358318
```

[5]:
```
7×2 Array{Float64,2}:
  -9.56448  -3.0
 -26.9833   -2.41331
 -28.203    -2.20513
 -28.2193   -2.17726
 -28.2193   -2.17677
 -28.2193   -2.17677
 -28.2193   -2.17677
```

[6]:
```
iter
```

[6]:
```
7×2 Array{Float64,2}:
  -9.56448  -3.0
 -26.9833   -2.41331
 -28.203    -2.20513
 -28.2193   -2.17726
 -28.2193   -2.17677
 -28.2193   -2.17677
 -28.2193   -2.17677
```

[7]:
```
plot(func, -3.2, -2.0, label="Function")
x = -3.0
m(y) = func(x)+dfunc(x)*(y-x)+0.5*d2func(x)*(y-x)^2
m(-3.0)
```

[7]:
```
-9.564480032239473
```

[8]:
```
plot!(m, -3.2, -2.0, label="Model")
plot!(iter[1:2,2], iter[1:2,1], label="Newton step")
vline!([iter[1,2] iter[2,2]], label = "")
```

[9]:
```
plot(func, -3.2, -2.0, label="Function")
plot!(iter[:,2], iter[:,1], label="Newton steps")
```

[10]:
```
x0 = -4.0
Newton(func, dfunc, d2func, x0, true)
```

```
1. x = -4.0, f(x) = 99.02720998123172
2. x = -2.942938833739946, f(x) = -12.3872911024757
3. x = -2.3879767853105314, f(x) = -27.24370765699775
4. x = -2.199836344922852, f(x) = -28.20853988049366
5. x = -2.177097516735148, f(x) = -28.219317146168752
6. x = -2.1767738416160567, f(x) = -28.21931925035823
7. x = -2.176773776419543, f(x) = -28.219319250358318
8. x = -2.1767737764195405, f(x) = -28.219319250358314
```

[11]:
```
x0 = -2.2
Newton(func, dfunc, d2func, x0, true)
```

```
1. x = -2.2, f(x) = -28.208385615278356
2. x = -2.177102076816536, f(x) = -28.219317086469566
3. x = -2.176773843465362, f(x) = -28.219319250358225
4. x = -2.1767737764195436, f(x) = -28.219319250358314
5. x = -2.176773776419541, f(x) = -28.219319250358318
```

However, if the method converges, it converges to a point where the derivative is equal to zero. This could be a local maximum!

[ ]:
```
x0 = 1.0
Newton(func, dfunc, d2func, x0, true)
```

[ ]:
```
  = 0.1
x = x0- *dfunc(x0)/d2func(x0)
  = 0.001
x = x0- *dfunc(x0)/d2func(x0)
```

[ ]:
```
f(x)
```

[ ]:
```
dfunc(x0)/d2func(x0)
```

[ ]:
```
dfunc(x0)
```

We can also converge to local, but not global, minimum.

[ ]:
```
x0 = 2.0
Newton(func, dfunc, d2func, x0, true)
```

1.2 Secant method

[ ]:
```
function Secant(f::Function, df::Function, x0::Float64, x1::Float64,
                verbose::Bool = false,  ::Float64 = 1e-6, nmax::Int64 = 1000)
    k = 1
    x = x0
    y = x1
    if (x0 == x1)
        println("x0 must different from x1")
        return
    end
    if (verbose)
        println("0. x0 = $x0, f($x0) = $(f(x0))")
        println("1. x1 = $x1, f($x1) = $(f(x1))")
    end
    dfx = df(x)
    dfy = df(y)
    while (abs(dfy) >   && k < nmax)
        k += 1
        t = y
        y = y-(x-y)/(dfx-dfy)*dfy
        x = t
        if (verbose)
            println("$k. x = $y, f(x) = $(f(y))")
        end
        dfx = dfy
        dfy = df(y)
    end

    return y
end
```

[ ]:
```
Secant(func, dfunc, 1.0, 2.0, true)
```

1.3 Root finding using Newton method

Iteration

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)}$$

Consider the function

[ ]:
```
f(x) = x-2sin(x)
```

[ ]:
```
df(x) = 1-2cos(x)
```

[ ]:
```
plot(f, -5.0, 5.0)
```

We are looking for a zero of $f(x)$.

[ ]:
```
function NewtonRoot(f::Function, df::Function, xstart::Float64,
        verbose::Bool = false,  ::Float64 = 1e-6, nmax::Int64 = 1000)
    k = 1
    x = xstart
    fx = f(x)
    if (verbose)
        println("$k. x = $x, f(x) = $fx")
    end
    while (abs(fx) >   && k < nmax)
        k += 1
        dfx = df(x)
        x = x-fx/df(x)
        fx = f(x)
        if (verbose)
            println("$k. x = $x, f(x) = $fx")
        end
    end

    return x
end
```

[ ]:
```
x0 = 1.1
NewtonRoot(f, df, x0, true)
```

It works, but we were close to a disaster! Observe that

[ ]:
```
df(1.1)
```

The curve is nearly flat, and we move to a distant point. Luckily we come back, but using many iterations. Consider now the starting point

[ ]:
```
x0 =  /3
```

The derivative at this point is

[ ]:
```
df(x0)
```

The Newton method now gives

[ ]:
```
NewtonRoot(f, df, x0, true)
```

We are less lucky! In fact, at $\pi/3$,

[ ]:
```
df( /3)
```

The Newton recurrence is under trouble as we have a division by zero. The function however proceeds as due to numerical errors, we have avoided the division by zero, but the method diverges, as $x \to -\infty$.

Take now a point with a significant derivative.

[ ]:
```
x0 = 4.0
NewtonRoot(f, df, x0, true)
```

The method now converges very quickly, even if the starting point was further from the solution.

Note the $x - 2\sin x = 0$ is equivalent to $\frac{1}{\sin x} - \frac{2}{x} = 0$ if we require that $x \neq k\pi$, $k \in \mathcal{Z}$. The function shape is nevertheless quite different around the function zero.

[ ]:
```
g(x) = 1/sin(x) - 2/x
plot(g, 0.1,  -0.1)
```

The derivative is

[ ]:
```
dg(x) = 2/(x*x)-cos(x)/(sin(x)^2)
```

[ ]:
```
x0 = 1.1
NewtonRoot(g, dg, x0, true)
```

We now observe a fast convergence, mainly due to the fact that the function does not present flat parts.

[ ]:
```
f(x) = exp(x/2)-x-1
```

[ ]:
```
df(x) = 0.5*exp(x/2)-1
```

[ ]:
```
x0 = 1.0
NewtonRoot(f, df, x0, true)
```

[ ]:
```
using Roots
```

[ ]:
```
y = fzero(df,0.0, 5.0)
```

[ ]:
```
x0 = y
NewtonRoot(f, df, x0, true)
```

1.4 Cycles

Consider now the function

```
[ ]: h(x) = x^3 - 2*x + 2
```

```
[ ]: plot(h, -2.5, 1.5)
```

```
[ ]: dh(x) = 3x^2-2
```

```
[ ]: x0 = 0.0
     NewtonRoot(h, dh, x0, true)
```

```
[ ]: ratio(x) = h(x)/dh(x)
```

```
[ ]: ratio(0.0)
```

```
[ ]: ratio(1.0)
```

The method cycles! However, if we change the starting point, we can converge.

```
[ ]: x0 = 0.5
     NewtonRoot(h, dh, x0, true)
```

1.5 Application: computation of the square root of a positive number.

The square root of a real number can be computed using the Newton method. More precisely, if we look for the square root of s, we can rewrite the problem as the computation of a zero of the function

$$f(x) = x^2 - s$$

Indeed, if $f(x) = 0$, then $s = x^2$ or $x = \sqrt{s}$. Develop the Newton recurrence for this problem.

Choosing a good starting point can however be an issue. (Wikipedia) With s expressed in scientific notation as $a \times 10^{2n}$ where $1 \leq a < 100$ and n is an integer, $\sqrt{s} \approx \sqrt{a}10^n$. The starting point is often computed as

$$\sqrt{s} = \begin{cases} 2 \times 10^n & \text{if } a < 10 \\ 6 \times 10^n & \text{if } a \geq 10 \end{cases}$$

The factors two and six are used because they approximate the geometric means of the lowest and highest possible values with the given number of digits:

$$\sqrt{\sqrt{1}\sqrt{10}} = 10^{\frac{1}{4}} \approx 2$$

and

$$\sqrt{\sqrt{10}\sqrt{100}} = 10^{\frac{3}{4}} \approx 6$$

```
[ ]: sq(x,s) = x*x-s
```

```
[ ]: dsq(x) = 2*x
```

[ ]:
```
sqs(x) = x*x-s
```

[ ]:
```
s = 6.0
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true)
```

We can check the solution as

[ ]:
```
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-8)
```

[ ]:
```
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-10)
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-12)
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-14)
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-16)
x^2
```

[ ]:
```
x = NewtonRoot(sqs, dsq, 2.0, true, 1e-15)
x^2
```

[ ]:
```
6-x^2
```

[ ]:
```
eps()
```

[ ]:
```
s = 25
x = NewtonRoot(sqs, dsq, 6.0, true, 1e-15)
x^2
```

[ ]:
```
s = 400
x = NewtonRoot(sqs, dsq, 6.0, true, 1e-15)
x^2
```

[ ]:
```

```