Simple_pendulum

June 18, 2020

This notebook uses a variety of different methods to solve the problem of the simple pendulum that starts at the positive x-axis with zero velocity. To be precise the differential equation being solved in this directory is:

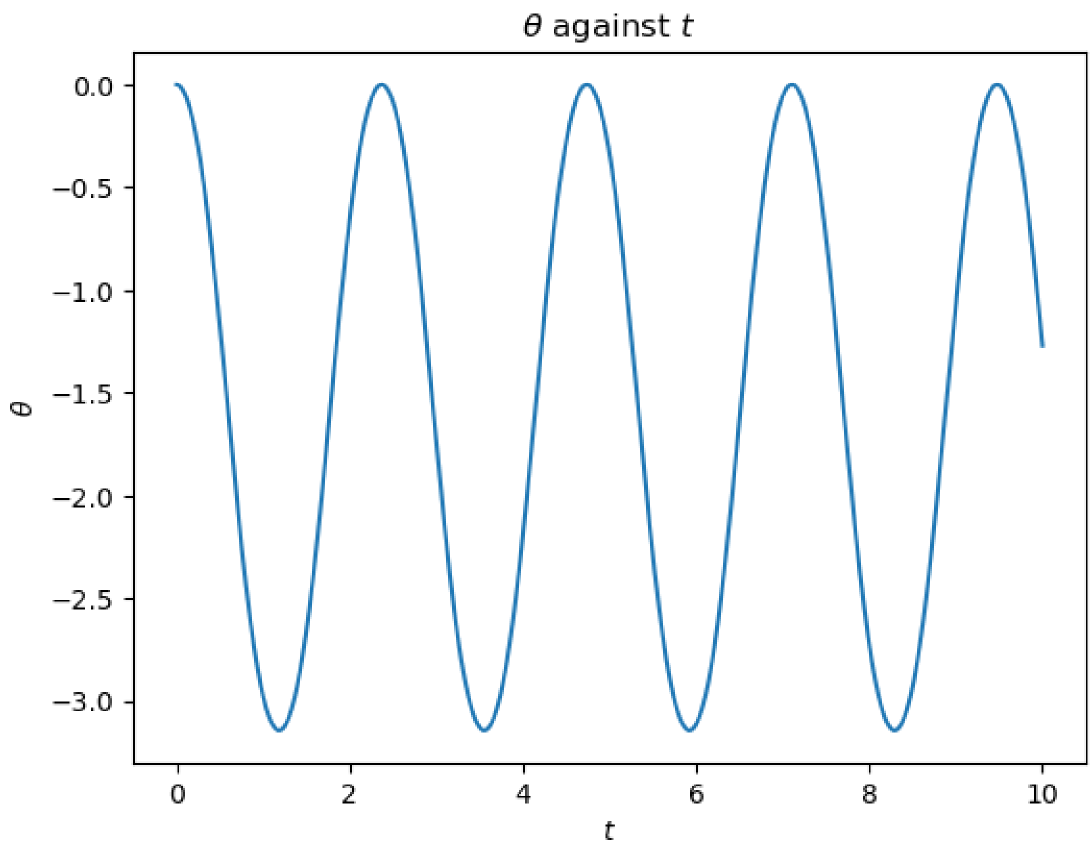
$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\cos\theta$$

subject to the initial conditions $\theta 0 = \dot{\theta}(0) = 0$, with $g = 9.8\text{m}\cdot\text{s}^{-2}$ and $l = 1.0\text{m}$.

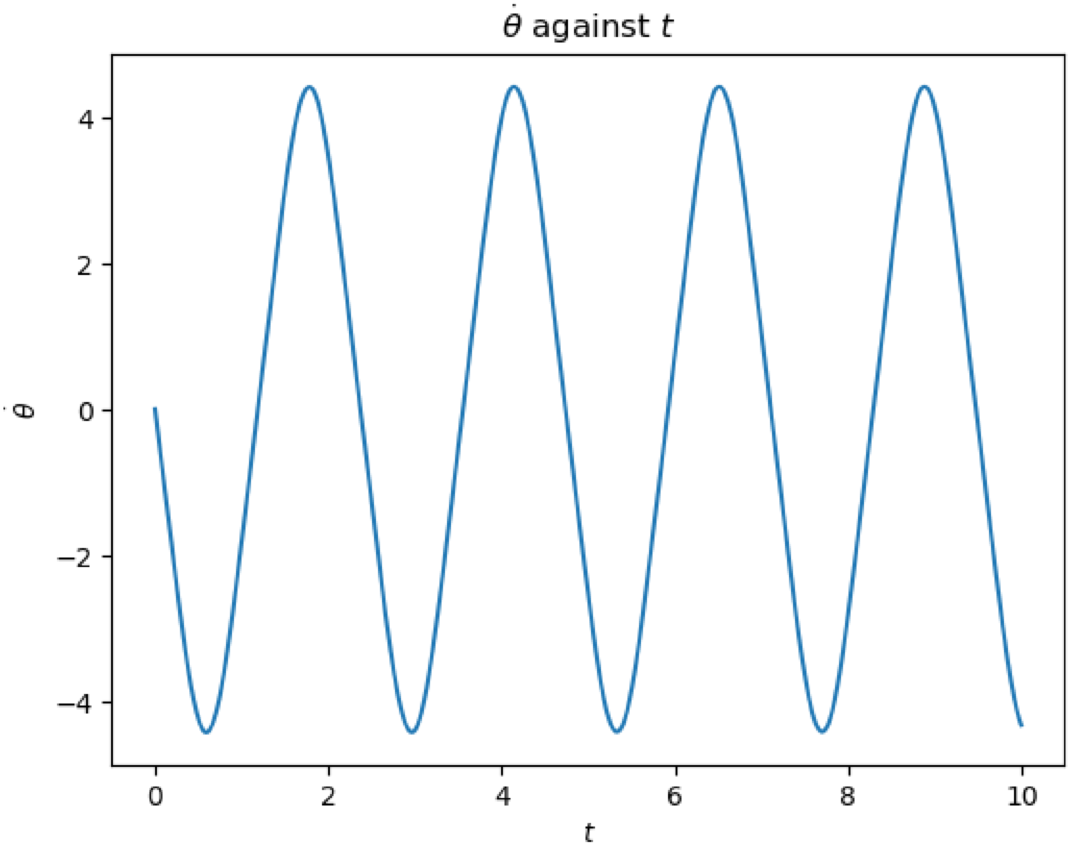
The first script uses the ODE.jl package's `ode78` solver.

[9]:
```
@time include("ode78.jl")
```

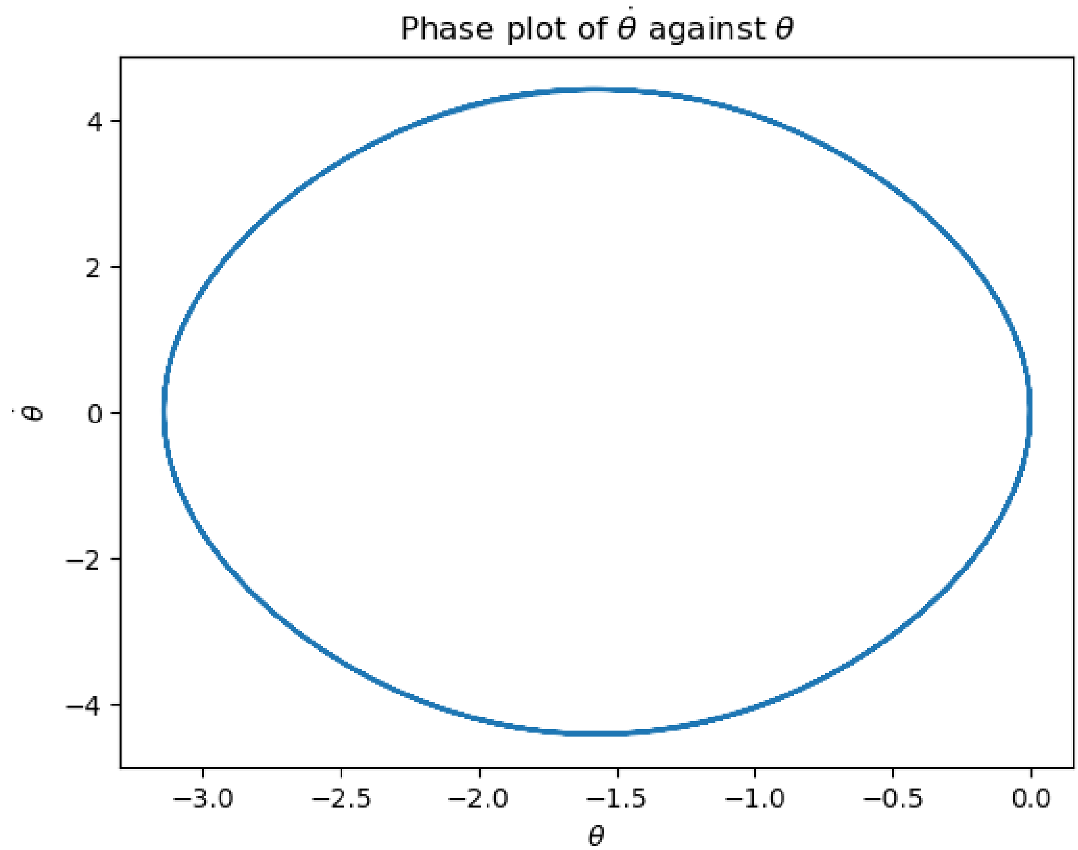
```
 Resolving package versions…
  Updating `~/.julia/environments/v1.4/Project.toml`
[no changes]
  Updating `~/.julia/environments/v1.4/Manifest.toml`
[no changes]
 Resolving package versions…
  Updating `~/.julia/environments/v1.4/Project.toml`
[no changes]
  Updating `~/.julia/environments/v1.4/Manifest.toml`
[no changes]
 Resolving package versions…
  Updating `~/.julia/environments/v1.4/Project.toml`
[no changes]
  Updating `~/.julia/environments/v1.4/Manifest.toml`
[no changes]
```

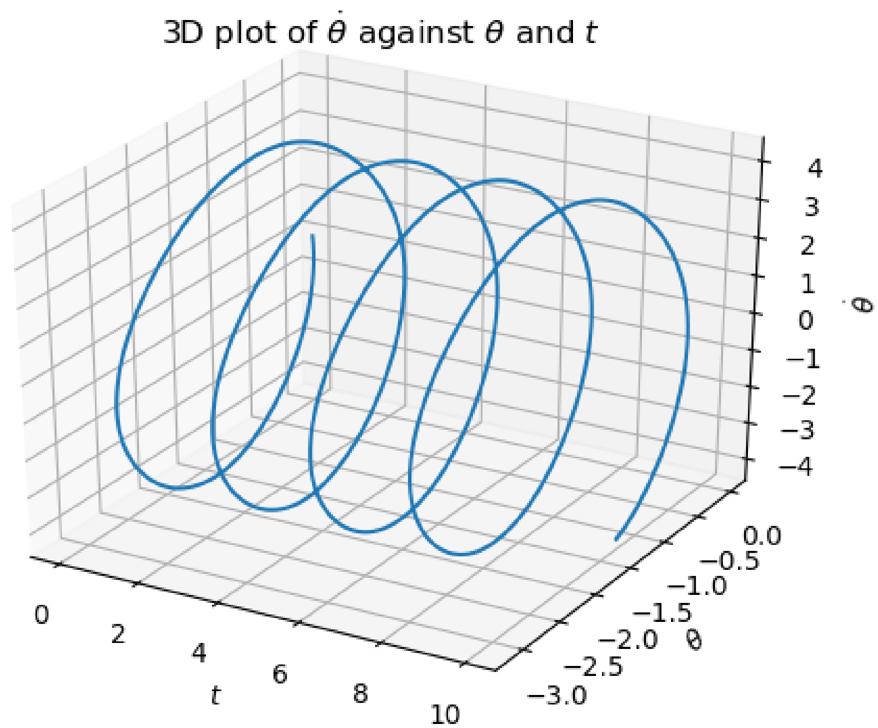


θ against t
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
θ
0
2
4
6
8
10
t



$\dot{\theta}$ against t
4
2
0
−2
−4
$\dot{\theta}$
0
2
4
6
8
10
t


Phase plot of $\dot{\theta}$ against θ
4
2
0
−2
−4
$\dot{\theta}$
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
θ




```
 25.787887 seconds (173.45 M allocations: 17.593 GiB, 30.50% gc time)
```

[9]: 490237154

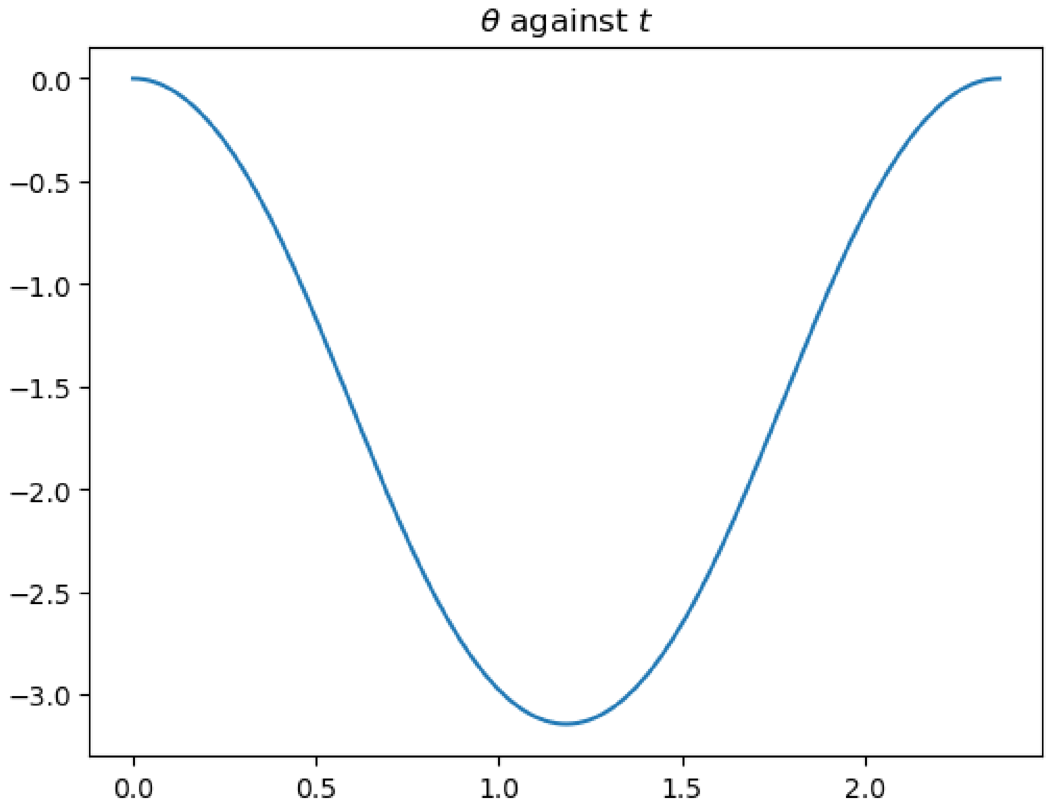
This next script uses the fourth-order Runge-Kutta method to approximate the solution to the problem.

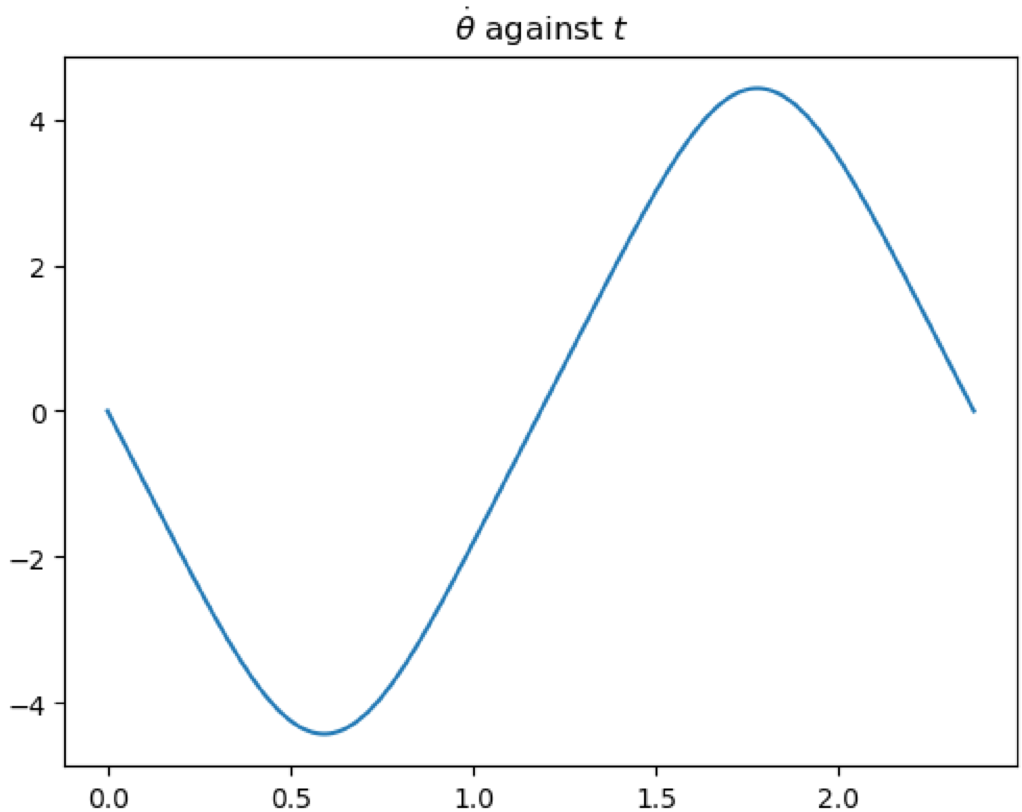
[12]:
```julia
@time include("RK4.jl")
```

```
  Resolving package versions…
   Updating `~/.julia/environments/v1.4/Project.toml`
 [no changes]
   Updating `~/.julia/environments/v1.4/Manifest.toml`
 [no changes]
  Resolving package versions…
```

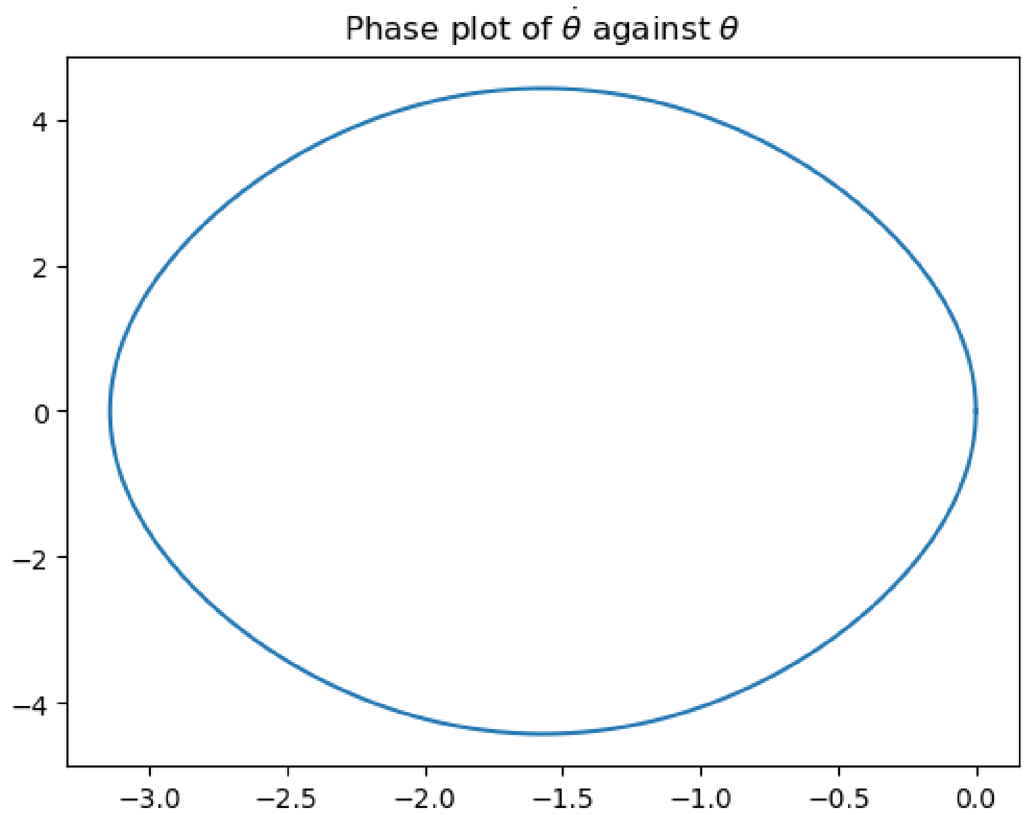
```
N is 10000000.
```

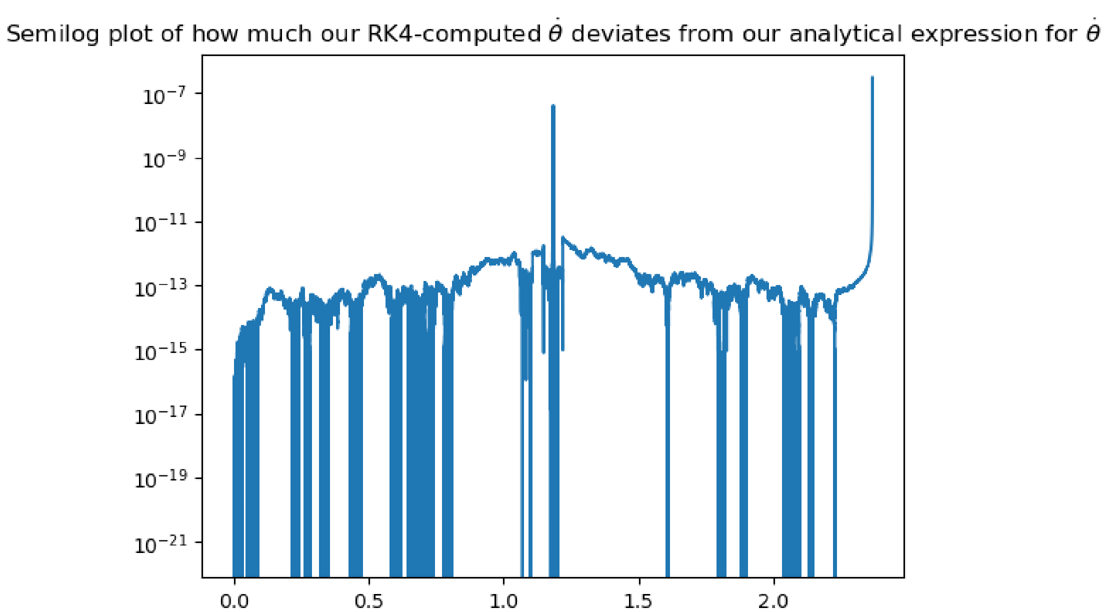
```
   Updating `~/.julia/environments/v1.4/Project.toml`
 [no changes]
   Updating `~/.julia/environments/v1.4/Manifest.toml`
 [no changes]
```


θ against t
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
0.0
0.5
1.0
1.5
2.0



$\dot{\theta}$ against t
4
2
0
−2
−4
0.0
0.5
1.0
1.5
2.0





`error_theta_min is 4.440892098500626e-16.`

```
error_dtheta_min is 7.993605777301127e-15.
rms_residual_dtheta is 9.789508631272697e-11.
 28.456525 seconds (731.58 M allocations: 11.800 GiB, 8.25% gc time)
```

And the final script uses Chebyshev spectral methods to approximate the solution to a linearized version of the ODE that was created using the Newton-Kantorovich method. Namely:

$$\ddot{\Delta}_i - \frac{g}{l}\sin\theta_i\Delta_i = -\ddot{\theta}_i - \frac{g}{l}\cos\theta_i$$

where $\theta_{i+1} = \theta_i + \Delta_i$, $\theta_0 = \frac{\pi}{2}\left(\cos\left(\frac{2\pi t}{\chi}\right) - 1\right)$ and χ is the period of the problem and is equal to:

$$2\int_{-\pi}^{0} \frac{d\theta}{\sqrt{-\frac{2g}{l}\sin\theta}}.$$

To approximate the solution of this integral, the script uses Chebyshev-Gauss quadrature. Namely, the approximation:

$$2\int_{-\pi}^{0} \frac{d\theta}{\sqrt{-\frac{2g}{l}\sin\theta}} \approx \frac{\pi^2}{N}\sqrt{\frac{l}{2g}}\sum_{i=1}^{N}\sqrt{\frac{1-x_i^2}{\cos\frac{\pi}{2}x_i}}$$

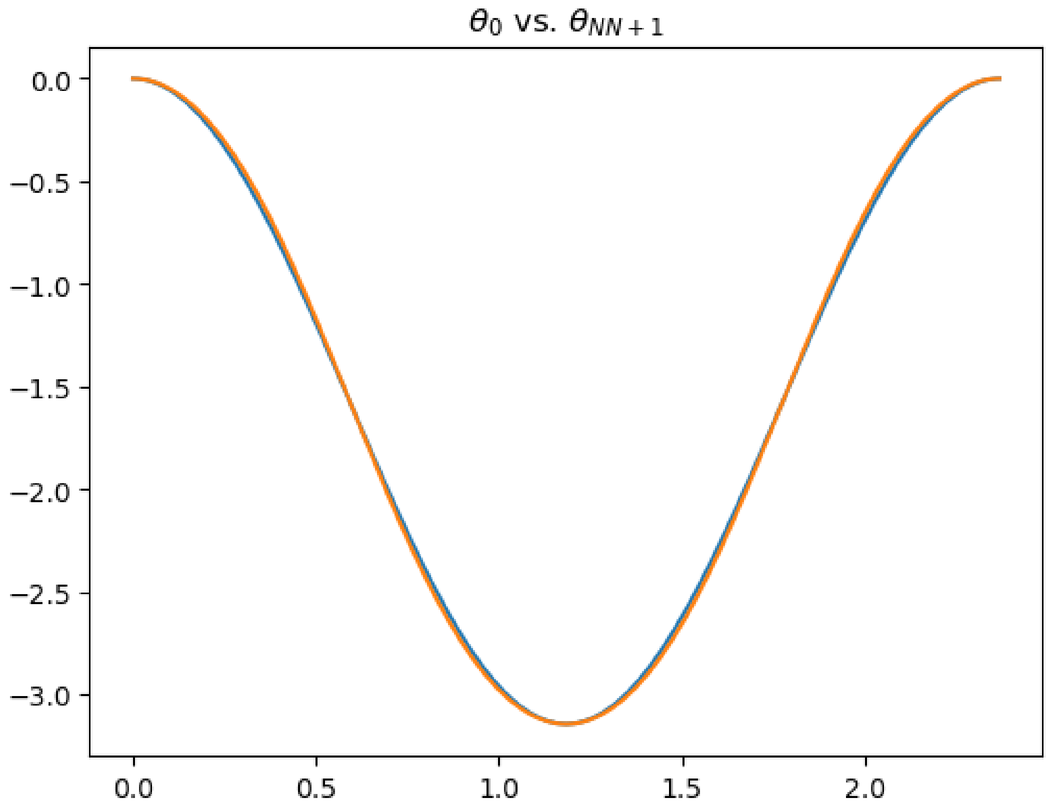
where $x_i = \cos\left(\frac{2i-1}{2N}\pi\right)$.

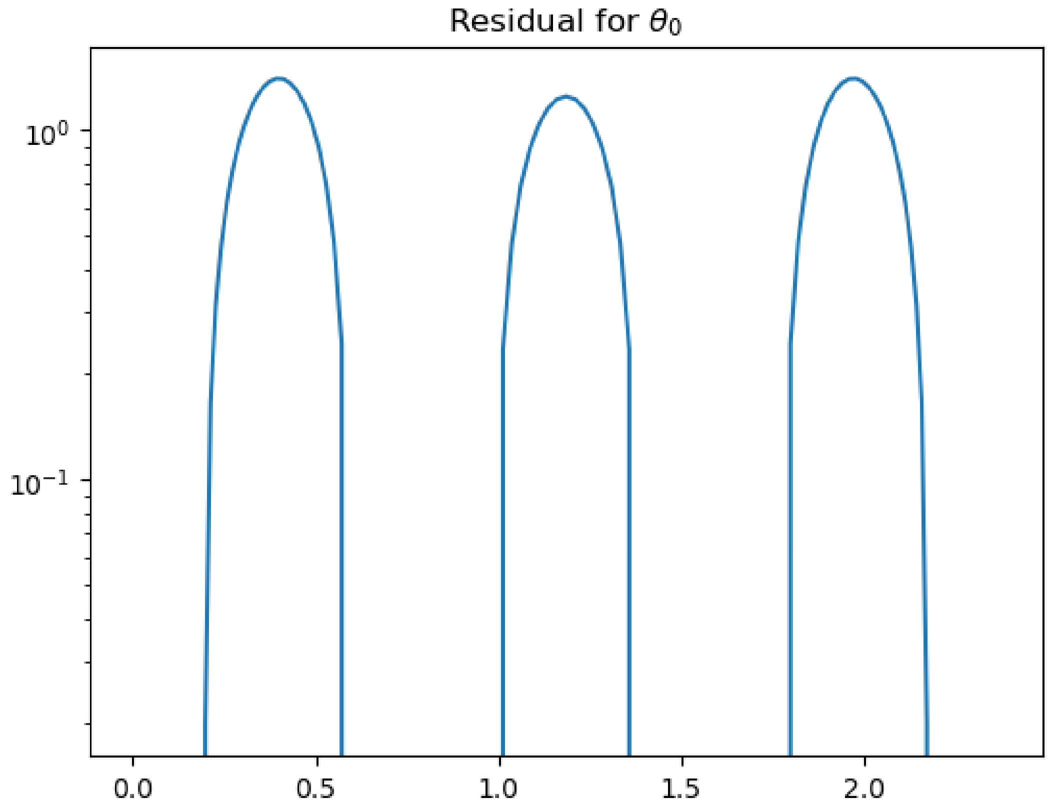
[15]:
```julia
@time include("Newton_Kantorovich_method.jl")
```

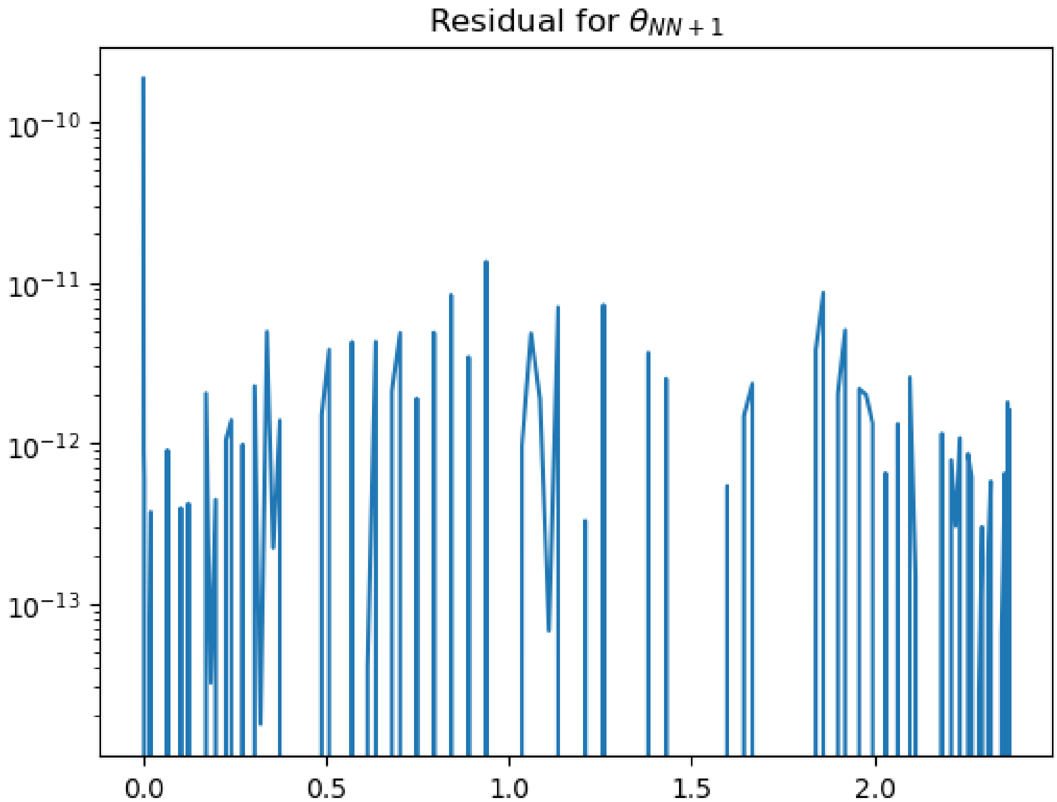
```
  Resolving package versions…
   Updating `~/.julia/environments/v1.4/Project.toml`
 [no changes]
   Updating `~/.julia/environments/v1.4/Manifest.toml`
 [no changes]
  Resolving package versions…
```

```
N is 150
NN is 4
period is 2.369049722175316.
```

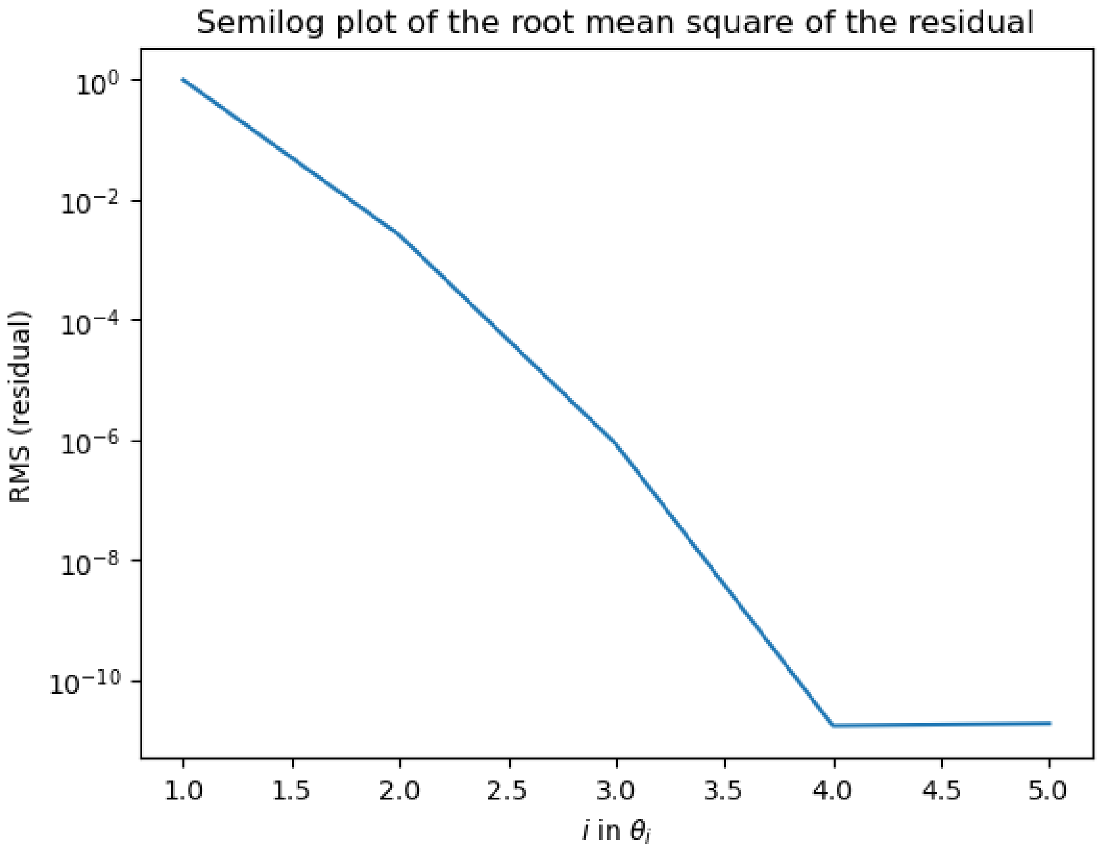
```
   Updating `~/.julia/environments/v1.4/Project.toml`
 [no changes]
   Updating `~/.julia/environments/v1.4/Manifest.toml`
 [no changes]
```


θ_0 vs. θ_{NN+1}
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
0.0
0.5
1.0
1.5
2.0


Residual for θ_0
10^0
10^{-1}
0.0
0.5
1.0
1.5
2.0



Residual for θ_{NN+1}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
0.0
0.5
1.0
1.5
2.0




```
rms_residual[1] is 0.9954579076617062
rms_residual[2] is 0.0025506985465578724
rms_residual[3] is 8.353465748873541e-7
rms_residual[4] is 1.714194443686345e-11
rms_residual[5] is 1.8705459885940126e-11
  0.943791 seconds (2.18 M allocations: 390.661 MiB, 5.87% gc time)
```

[15]: PyObject Text(0.5, 1.0, 'Semilog plot of the root mean square of the residual')

[ ]: