Exercise 1

In [1]:
```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# define functions given in the exercise
def f(x):
    return (x**3 +1/3)-(x**3 - 1/3)

def g(x):
    return ((3+x**(3)/3)-(3-x**(3)/3))/x**3
```

a)

In [3]:
```
# define range of x values
n = 1000000000000000000000
```

In [4]:
```
# print x value where the numerical solution deviates by 1% from the algebraic
for x in range(n):
    if(np.abs((f(x)-2/3)/(2/3)) > 0.01):
        print('The numerical solution starts to deviate from the algebraic solutio
        break

for x in range(n):
    if(np.abs((f(-x)-2/3)/(2/3)) > 0.01):
        print('The numerical solution starts to deviate from the algebraic solutio
        break
```

```
The numerical solution starts to deviate from the algebraic solution at x >=  4128
6
The numerical solution starts to deviate from the algebraic solution at x =<  4128
6
```

In [5]:
```
# print x value where the numerical solution equals 0
for x in range(n):
    if(f(x)==0):
        print('The numerical solution is equal to zero at x >= ', x)
        break

for x in range(n):
    if(f(-x)==0):
        print('The numerical solution is equal to zero at x =< ', x)
        break
```

```
The numerical solution is equal to zero at x >=  165141
The numerical solution is equal to zero at x =<  165141
```

b)

In [6]:
```
# print x value where the numerical solution deviates by 1% from the algebraic
for x in range(1, n):
    if(np.abs((g(1/x)-2/3)/(2/3)) > 0.01):
        print('The numerical solution starts to deviate from the algebraic solutio
        break
```

```
for x in range(1, n):
    if(np.abs((g(-1/x)-2/3)/(2/3)) > 0.01):
        print('The numerical solution starts to deviate from the algebraic solutio
        break
```

```
The numerical solution starts to deviate from the algebraic solution at x >=  2475
1
The numerical solution starts to deviate from the algebraic solution at x =<  2475
1
```

In [7]:

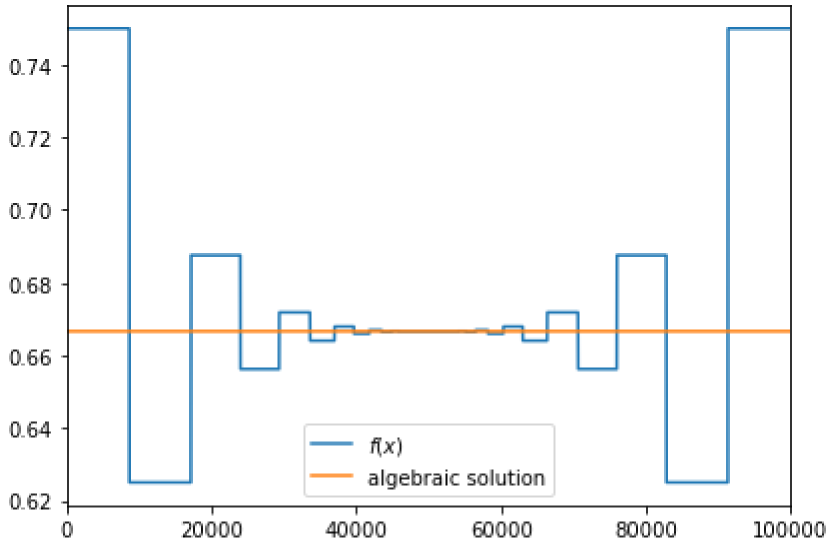
```
# print x value where the numerical solution equals 0
for x in range(1, n):
    if(g(1/x)==0):
        print('The numerical solution is equal to zero at x >= ', 1/x)
        break

for x in range(1, n):
    if(g(-1/x)==0):
        print('The numerical solution is equal to zero at x =< ', 1/x)
        break
```

```
The numerical solution is equal to zero at x >=  8.733471904420884e-06
The numerical solution is equal to zero at x =<  8.733471904420884e-06
```

c)

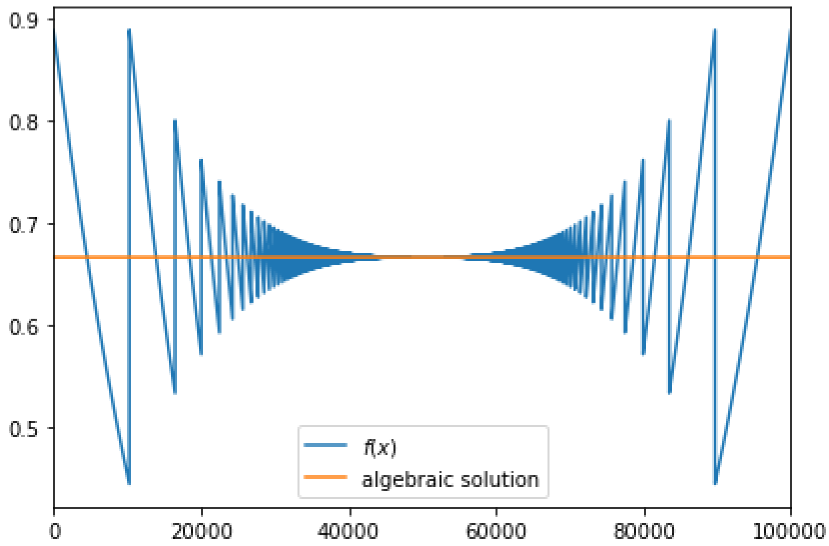
In [8]:

```
# Plot f(x)
x = np.linspace(-100000, 100000, 100000)
plt.plot(f(x), label=r'$f(x)$')
plt.plot(x, 0*x + 2/3, label='algebraic solution')
plt.xlim(0, 100000)
plt.legend()
plt.tight_layout()
plt.show()
```



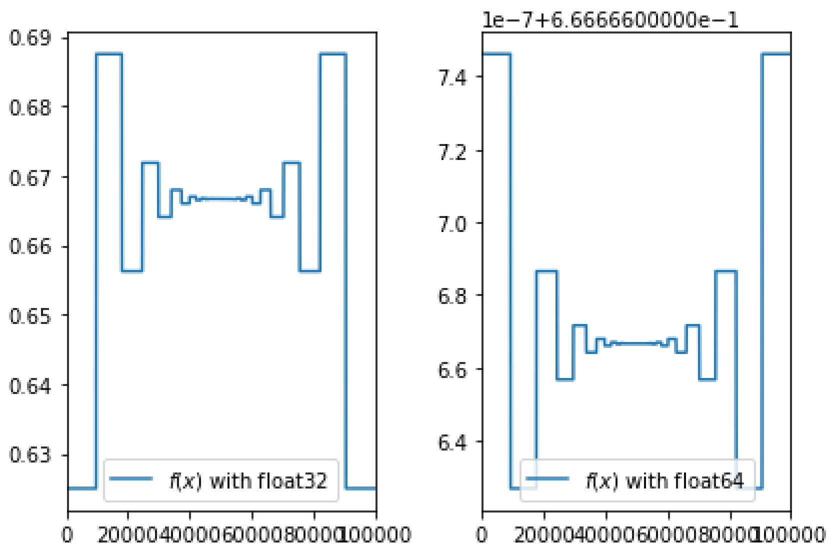
In [9]:

```
# Plot g(x)
x = np.linspace(-100000, 100000, 100000)
plt.plot(g(1/x), label=r'$f(x)$')
plt.plot(x, 0*x + 2/3, label='algebraic solution')
plt.xlim(0, 100000)
plt.legend()
plt.tight_layout()
plt.show()
```



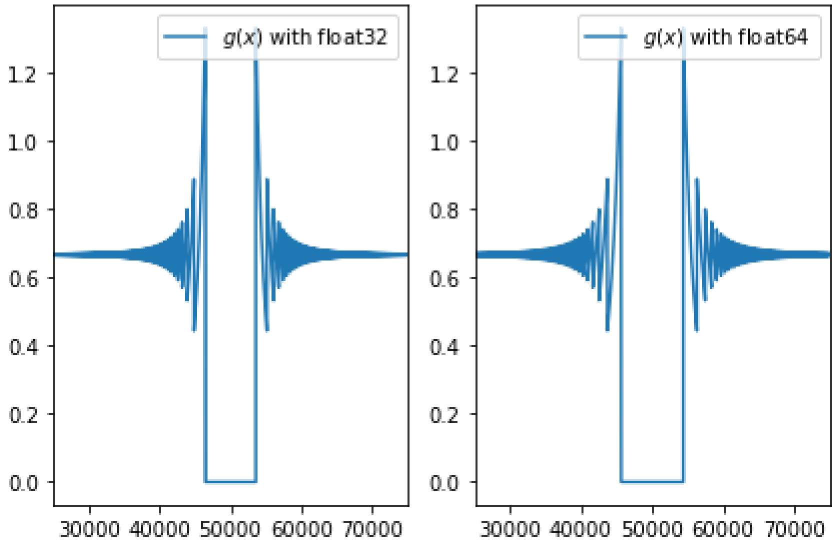
d)

In [10]:
```
# Plot f(x)
x32 = np.linspace(-100, 100, 100000, dtype='float32')
x64 = np.linspace(-1000, 1000, 100000, dtype='float64')

plt.subplot(1, 2, 1)
plt.plot(f(x32), label=r'$f(x)$ with float32')
plt.xlim(0, 100000)
plt.legend(loc='lower center')
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.plot(f(x64), label=r'$f(x)$ with float64')
plt.xlim(0, 100000)
plt.legend(loc='lower center')
plt.tight_layout()
plt.show()
```



In [11]:
```
# Plot g(x)
x32 = np.linspace(-0.1, 0.1, 100000, dtype='float32')
x64 = np.linspace(-0.0001, 0.0001, 100000, dtype='float64')

plt.subplot(1, 2, 1)
plt.plot(g(x32), label=r'$g(x)$ with float32')
plt.xlim(25000, 75000)
```

```
plt.legend()
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.plot(g(x64), label=r'$g(x)$ with float64')
plt.xlim(25000, 75000)
plt.legend()
plt.tight_layout()
plt.show()
```



The float32 plot shows less numerical stable values than the float64 plot.

Exercise 2

a)

The function is numerically unstable for $\theta \to 0$. This is true for $E \neq 0$.

In [12]:
```
# define mass
m = 511e3

#define functions
def gamma(E):
    return E/m

def beta(E):
    return np.sqrt(1-gamma(E)**(-2))

def f(E, theta):
    return (2+np.sin(theta)**2)/(1-(beta(E)**2)*(np.cos(theta)**2))
```

b)

$\frac{2+\sin^2\theta}{1-\beta^2\cos^2\theta}$ can be transformed to

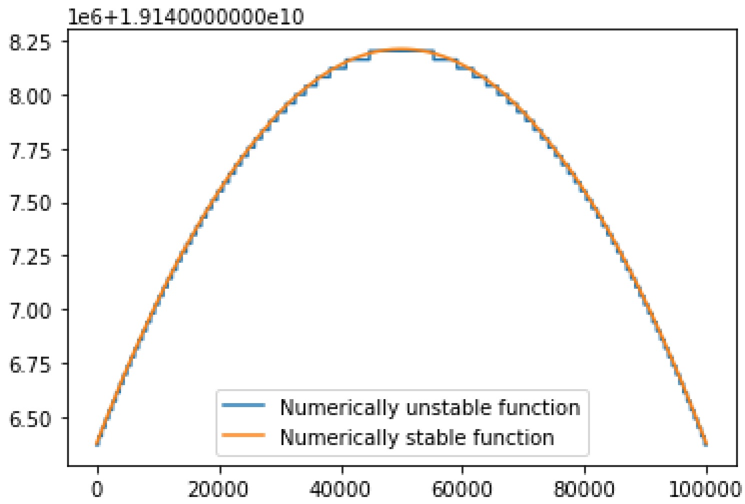
$\frac{2+sin^2\theta}{\frac{1}{\gamma^2}+(\beta sin\theta)^2}$ through using

$cos^2\theta = 1 - sin^2\theta$ and $\frac{1}{\gamma^2} = 1 - \beta^2$.

In [13]:
```
# define new function
def f_new(E, theta):
    return (2+np.sin(theta)**2)/(gamma(E)**(-2)+(beta(E)*np.sin(theta))**2)
```

c)

In [14]:
```
# plotting both functions in the critical interval
x = np.linspace(-1e-7, 1e-7, 100000, dtype='float64')
plt.plot(f(50e9, x), label='Numerically unstable function')
plt.plot(f_new(50e9, x), label='Numerically stable function')
plt.legend()
plt.show()
```



d)

The condition number is defined as,

$$K \equiv \left|\theta \cdot \frac{f'(\theta)}{f(\theta)}\right|.$$

With $f(\theta)$ being the numerically stable version and

$$f'(\theta) = \frac{2\cos(\theta)\left(\frac{1}{\gamma^2} + \beta^2\sin^2(\theta)\right) - \left(2+\sin^2(\theta)\right)\left(2\beta^2\cos(\theta)\right)}{\left(\frac{1}{\gamma^2} + \beta^2\sin^2(\theta)\right)}.$$

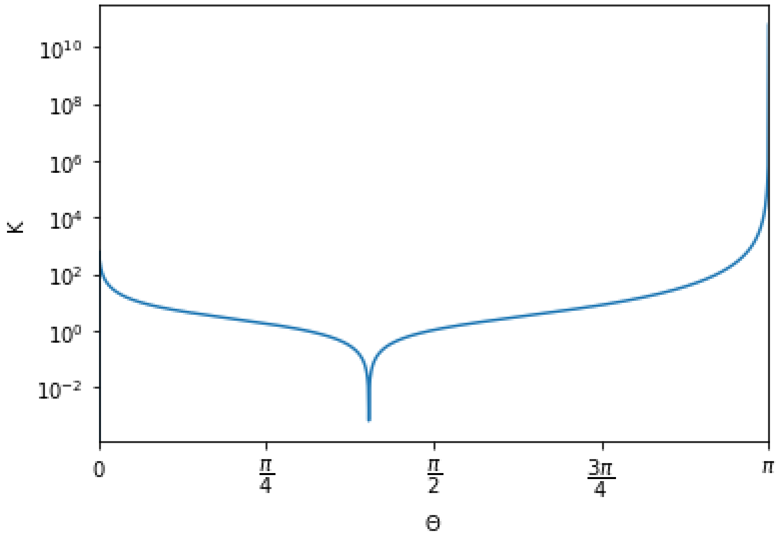
Finally the condition number is given as,

$$K = \left|\theta\frac{f'(\theta)}{f(\theta)}\right| = \left|\frac{2\theta\cos(\theta)}{2+\sin^2(\theta)} - \frac{2\theta\beta^2\cos(\theta)}{\frac{1}{\gamma^2} + (\beta\sin(\theta))^2}\right|.$$

e)

In [15]:

```
# define the condition number in code
def cond_numb(E, theta):
    return abs((2*theta*np.sin(theta))/(2+np.sin(theta)**2) - (2*theta * beta(E)**

x = np.linspace(0, np.pi, 1000)
plt.plot(x, cond_numb(50e9, x), label="condition number for $E=50$ GeV")

# make the plot look more beautiful
plt.xticks(np.arange(0, np.pi+0.001, step=np.pi/4), [r'$0$',r'$\dfrac{\pi}{4}$', r
plt.xlim(0, np.pi)
plt.yscale('log')
plt.xlabel(r'$\Theta$')
plt.ylabel('K')
plt.show()
```



f)

Stability is the influence of rounding errors for *inexact* computation, while **condition** is the propagation of initial uncertainties for *exact* comptation. To be more precise, the difference between stability and condition is that one describes the error for inexact (stability) and one the error for exact (condition) cumputation.

In [ ]: