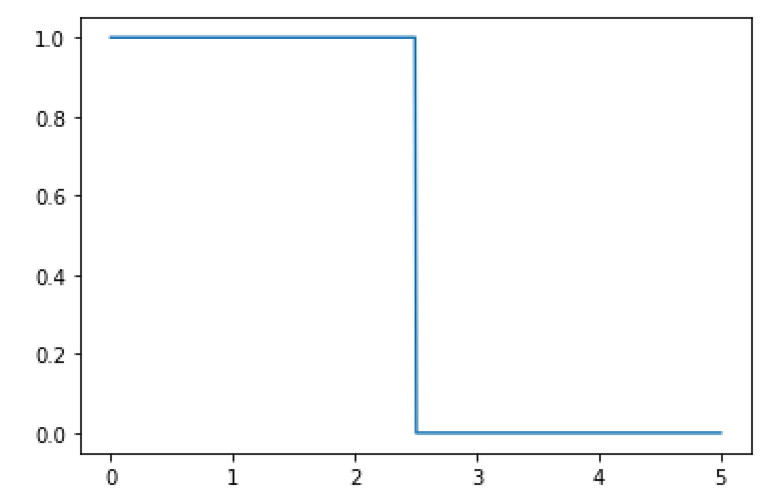
Parabolic PDE's

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.animation as ani
```

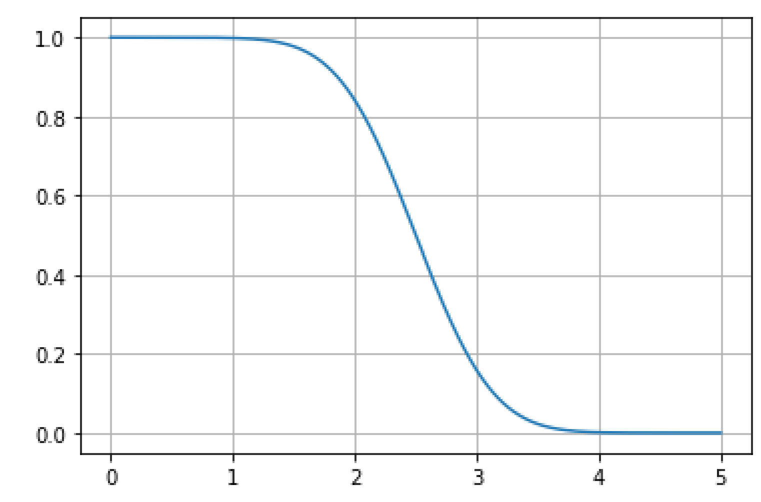
In [4]:
```
L = 5
B = 3
N = 500
x = np.linspace(0, L, N)
dx = x[1] - x[0]
dt = dx**2/4
y = np.where((x <= L/2), 1., 0.)
D = np.zeros([10*N,N],np.float)
D[0:] = y
```

In [5]:
```
plt.plot(x, y)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1c5654089a0>]
```



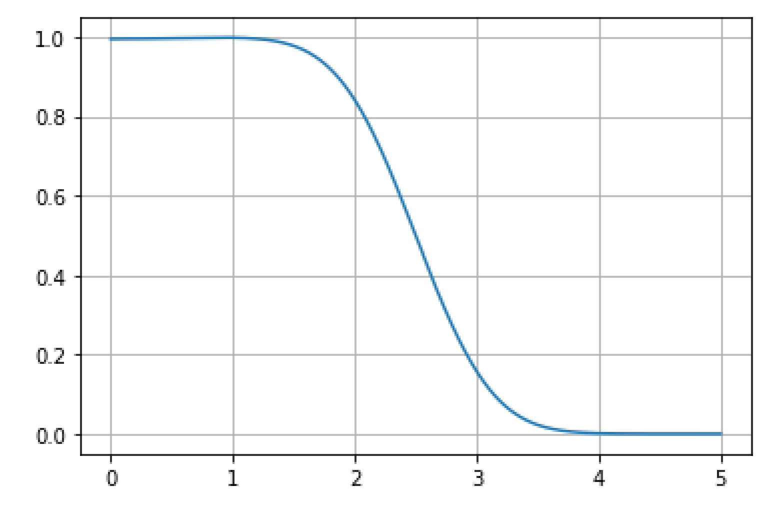
In [20]:
```
for i in range(1,10*N-1):
    for j in range(1,N-1):
        D[i+1, j] = D[i, j] + dt/dx**2*(D[i, j+1] - 2*D[i, j] + D[i, j-1])
    D[i+1, 0] = D[i, 0] + dt/dx**2*(D[i, 1] - 2*D[i, 0] + D[i, 1])
    D[i+1, N-1] = D[i, N-1] + dt/dx**2*(D[i, N-2] - 2*D[i, N-1] + D[i, N-2])

plt.plot(x, D[i+1])
plt.grid()
```



In [21]:
```
t = 0

def fa_vec(x):

    return np.where((x <= L/2), 1., 0.)

def basis(x, n, t):
    k = n*np.pi/L
    if n == 0:
        return np.sqrt(1/L)*np.ones(len(x))
    else:
        base = np.sqrt(2/L)*np.cos(k*x) * np.exp(-(k**2)*t)
        return base
```

In [22]:
```
def simpson_array(f, h):
    """

Use Simpson's Rule to estimate an integral of an array of
    function samples

    f: function samples (already in an array format)
    h: spacing in "x" between sample points

    The array is assumed to have an even number of elements.

    """
    if len(f)%2 != 0:
        raise ValueError("Sorry, f must be an array with an even number of elements.")

    evens =  f[2:-2:2]
    odds = f[1:-1:2]
    return (f[0] + f[-1] + 2*odds.sum() + 4*evens.sum())*dx/3.0

def braket(n,t):
    """
    Evaluate <n|f>
    """
    return simpson_array(basis(x,n,t)*fa_vec(x),dx)


M=100
coefs = []

for n in range(M):
    coefs.append(braket(n,0))   # do numerical integral

sup = np.zeros(N)

def Superposition(t):
    sup = np.zeros(N)
    for n in range(M):
        sup += coefs[n]*basis(x,n, t)
    return sup

T = 0.125

plt.plot(x,Superposition(T))
plt.grid()
print("%10s\t%10s" % ('n', 'coef'))
print("%10s\t%10s" % ('---','-----'))
for n in range(M):
    print("%10d\t%10.5f" % (n, coefs[n]))
```

```
         n            coef
       ---           -----
         0         1.11579
         1         1.00447
         2        -0.00106
         3        -0.33765
         4        -0.00317
         5         0.19921
         6        -0.00106
         7        -0.14592
         8        -0.00317
         9         0.10975
        10        -0.00106
        11        -0.09364
        12        -0.00317
        13         0.07534
        14        -0.00106
        15        -0.06924
        16        -0.00317
        17         0.05713
        18        -0.00105
        19        -0.05512
        20        -0.00317
        21         0.04586
        22        -0.00105
        23        -0.04592
        24        -0.00317
        25         0.03819
        26        -0.00105
        27        -0.03944
        28        -0.00317
        29         0.03265
        30        -0.00105
        31        -0.03463
        32        -0.00317
        33         0.02845
        34        -0.00105
        35        -0.03093
        36        -0.00318
        37         0.02515
        38        -0.00105
        39        -0.02799
        40        -0.00318
        41         0.02251
        42        -0.00105
        43        -0.02559
        44        -0.00318
        45         0.02033
        46        -0.00105
        47        -0.02361
        48        -0.00318
        49         0.01851
        50        -0.00104
        51        -0.02193
        52        -0.00318
        53         0.01697
        54        -0.00104
        55        -0.02051
        56        -0.00319
        57         0.01564
        58        -0.00104
        59        -0.01927
        60        -0.00319
        61         0.01449
        62        -0.00104
        63        -0.01820
        64        -0.00319
        65         0.01348
        66        -0.00103
        67        -0.01725
        68        -0.00319
        69         0.01259
        70        -0.00103
        71        -0.01641
        72        -0.00320
        73         0.01180
        74        -0.00103
        75        -0.01566
        76        -0.00320
        77         0.01109
        78        -0.00102
        79        -0.01499
        80        -0.00320
        81         0.01045
        82        -0.00102
        83        -0.01438
        84        -0.00321
        85         0.00987
        86        -0.00102
        87        -0.01383
        88        -0.00321
        89         0.00935
        90        -0.00101
        91        -0.01333
        92        -0.00321
        93         0.00887
        94        -0.00101
        95        -0.01287
        96        -0.00322
        97         0.00843
        98        -0.00100
        99        -0.01245
```



In [23]:
```
np.sqrt(((D[i+1] - Superposition(T))**2).sum()/len(x))
```

Out[23]:
```
0.0017028881160125735
```