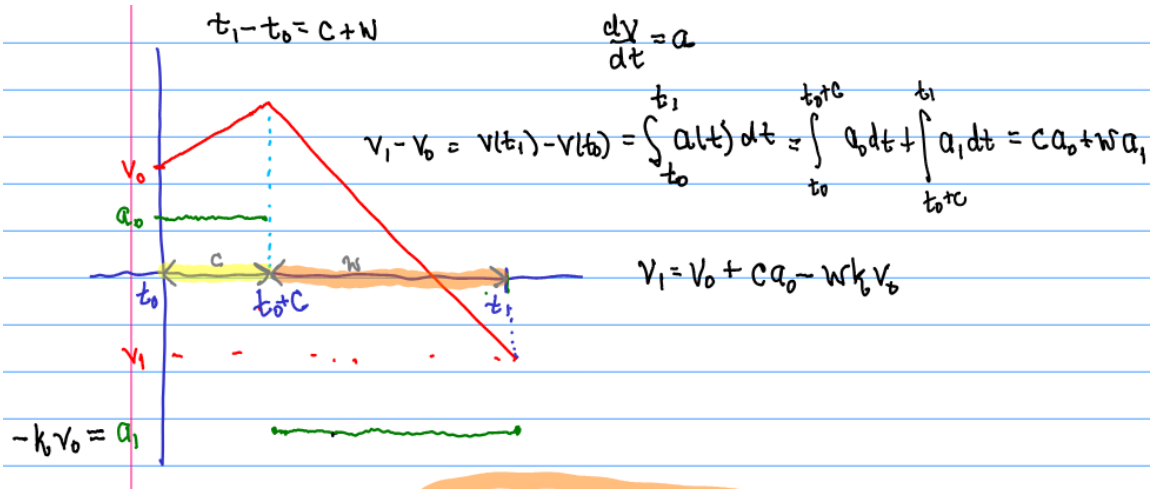
Example 4.3. Astronauts in training are required to practice a docking maneuver under manual control. As a part of this maneuver, it is required to bring an orbiting spacecraft to rest relative to another orbiting craft. The hand controls provide for variable acceleration and deceleration, and there is a device on board that measures the rate of closing between the two vehicles. The following strategy has been proposed for bringing the craft to rest. First, look at the closing velocity. If it is zero, we are done. Otherwise, remember the closing velocity and look at the acceleration control. Move the acceleration control so that it is opposite to the closing velocity (i.e., if closing velocity is positive, we slow down, and we speed up if it is negative) and proportional in magnitude (i.e., we brake twice as hard if we find ourselves closing twice as fast). After a time, look at the closing velocity again and repeat the procedure. Under what circumstances will this strategy be effective?



In Lecture 21 we arrive at the dynamical system

$$X_{n+1} = AX_n \quad \text{where} \quad A = \begin{bmatrix} 1 - w\kappa & -c\kappa \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad X_n = \begin{bmatrix} v_n \\ v_{n-1} \end{bmatrix}.$$

In [1]:
```
using Symbolics
```

In [2]:
```
@variables c,w,kappa
```

Out[2]:
```
3-element Vector{Num}:
     c
     w
 kappa
```

In [3]:
```
A=[1-w*kappa -c*kappa; 1 0]
```

Out[3]:
```
2×2 Matrix{Num}:
 1 - kappa*w  -c*kappa
           1         0
```

In [4]:
```
# Create a function of kappa returning the matrix A
fAs="fA(kappa)="*string(Symbolics.toexpr(A))
eval(Meta.parse(fAs))
```

Out[4]:
```
fA (generic function with 1 method)
```

In [5]:
```
fA(kappa)
```

Out[5]:
```
2×2 Matrix{Num}:
 1 - kappa*w  -c*kappa
           1         0
```

In [6]:
```
# Fix c and w so the effects of kappa can be explored
c=5
w=10
fA(kappa)
```

Out[6]:
```
2×2 Matrix{Num}:
 1 - 10kappa  -5kappa
           1        0
```

In [7]:
```
using LinearAlgebra
```

In [8]:
```
eigvals(fA(0.02))
```

Out[8]:
```
2-element Vector{Float64}:
 0.1550510257216822
 0.6449489742783179
```

For small positive values of κ the eigenvalues are both positive and less than 1.

In [9]:
```
eigvals(fA(0.03))
```

Out[9]:
```
2-element Vector{ComplexF64}:
 0.35 - 0.16583123951777004im
 0.35 + 0.16583123951777004im
```

In [10]:
```
abs.(eigvals(fA(0.03)))
```

Out[10]:
```
2-element Vector{Float64}:
 0.3872983346207417
 0.3872983346207417
```

When κ is a little larger the eigenvalues are complex and have equal magnitudes which are still less than 1.

In [11]:
```
eigvals(fA(0.3))
```

Out[11]:
```
2-element Vector{ComplexF64}:
 -1.0 - 0.7071067811865475im
 -1.0 + 0.7071067811865475im
```

When κ is a bit greater still the eigenvalues are complex but have magnitudes greater than 1.

In [12]:
```
eigvals(fA(0.5))
```

Out[12]:
```
2-element Vector{Float64}:
 -3.2247448713915894
 -0.7752551286084108
```

Whe κ is much larger the eigenvalues are negative and one of them has a magnitude much greater than 1.

In [13]:
```
# Create functions to plot the magnitude of the eigenvalues
fabseig1(kappa)=abs(eigvals(fA(kappa))[1])
fabseig2(kappa)=abs(eigvals(fA(kappa))[2])
```

Out[13]:
```
fabseig2 (generic function with 1 method)
```

In [14]:
```
fabseig1(0.5)
```

Out[14]:
```
3.2247448713915894
```

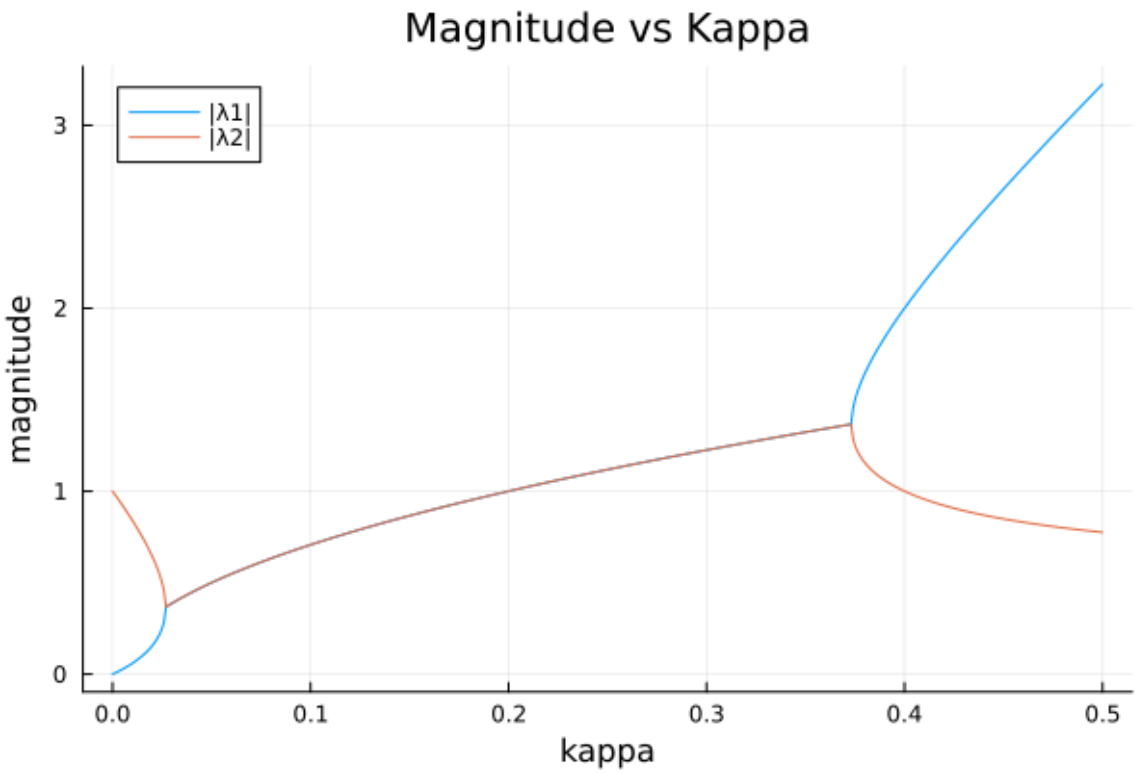
In [15]:
```
fabseig2(0.5)
```

Out[15]:
```
0.7752551286084108
```

In [16]:
```
using Plots
```

In [17]:
```
plot([fabseig1,fabseig2],0:0.001:0.5,
    xlabel="kappa",ylabel="magnitude",
    title="Magnitude vs Kappa",
    label=["|λ1|" "|λ2|"])
```

Out[17]:



Note the κ where $\max(|\lambda_1|, |\lambda_2|)$ is minimized happens when the discriminate is zero--just before the eigenvalues turn complex. Then the eigenvalues remain less than 1 until about $\kappa = 0.2$. After this at least one of the eigenvalues has magnitude greater than 1.

In [18]:
```
# The eigenvalues with no control at all
eigvals(fA(0.0))
```

Out[18]:
```
2-element Vector{Float64}:
 0.0
 1.0
```

In [ ]:
```

```