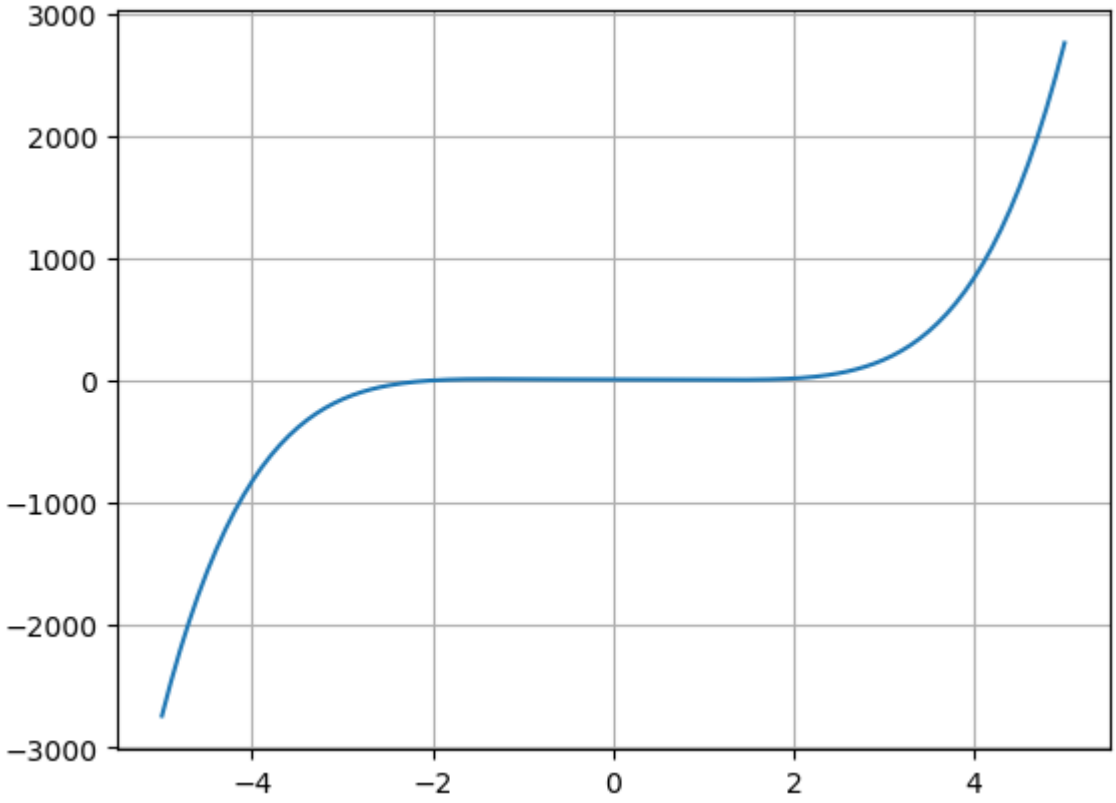
In [36]:
```
#Ivan Wang #50414321
```

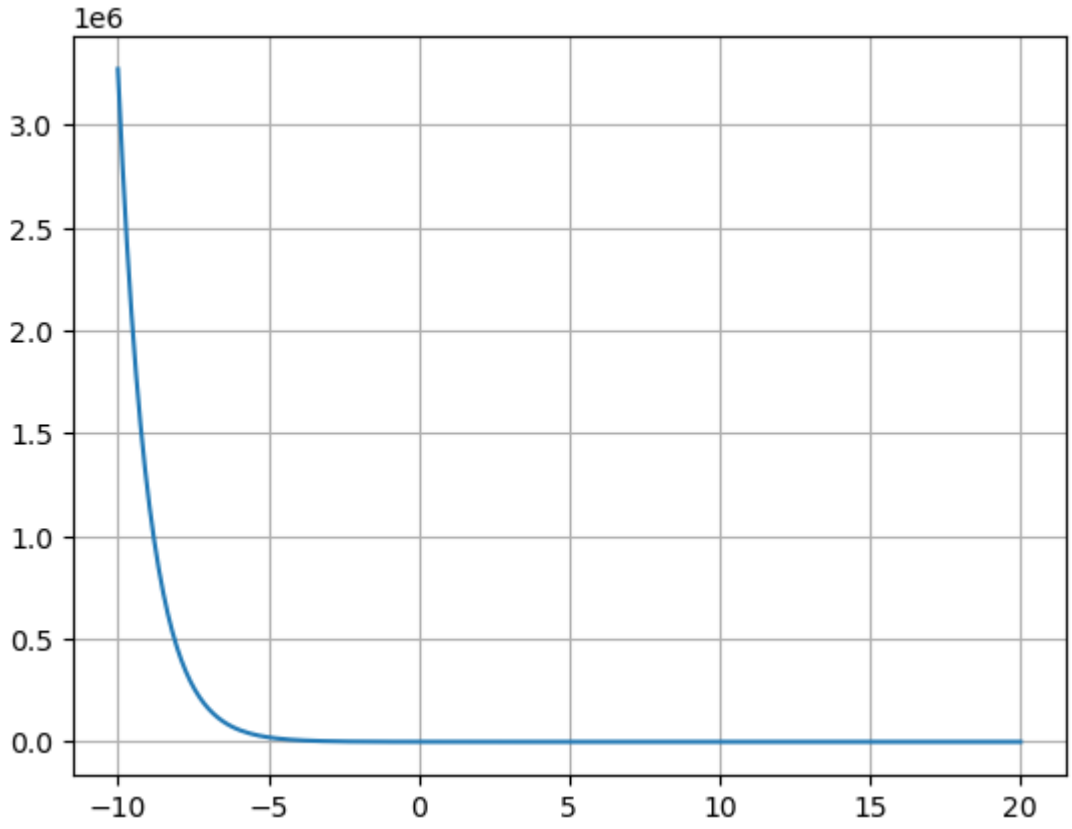
In [37]:
```
#Pizza #Dire Straits
```

In [38]:
```
import requests
with open("resources306.py", "w") as f:
    f.write(requests.get("https://cutt.ly/mth306").text)
```

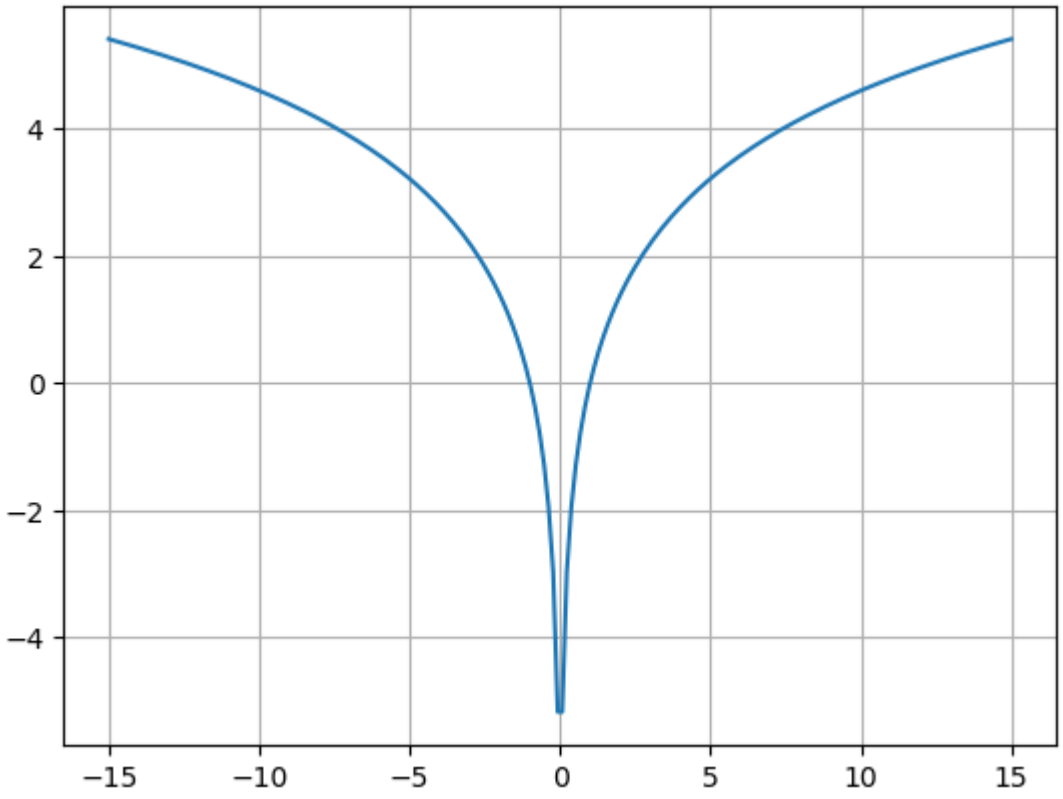
In [39]:
```
import sympy as sp, numpy as np, matplotlib.pyplot as plt
from resources306 import *

t = np.linspace(-5,5,200)

plt.plot( t, t**5 - 3*t**3 + 5);

plt.grid()
```



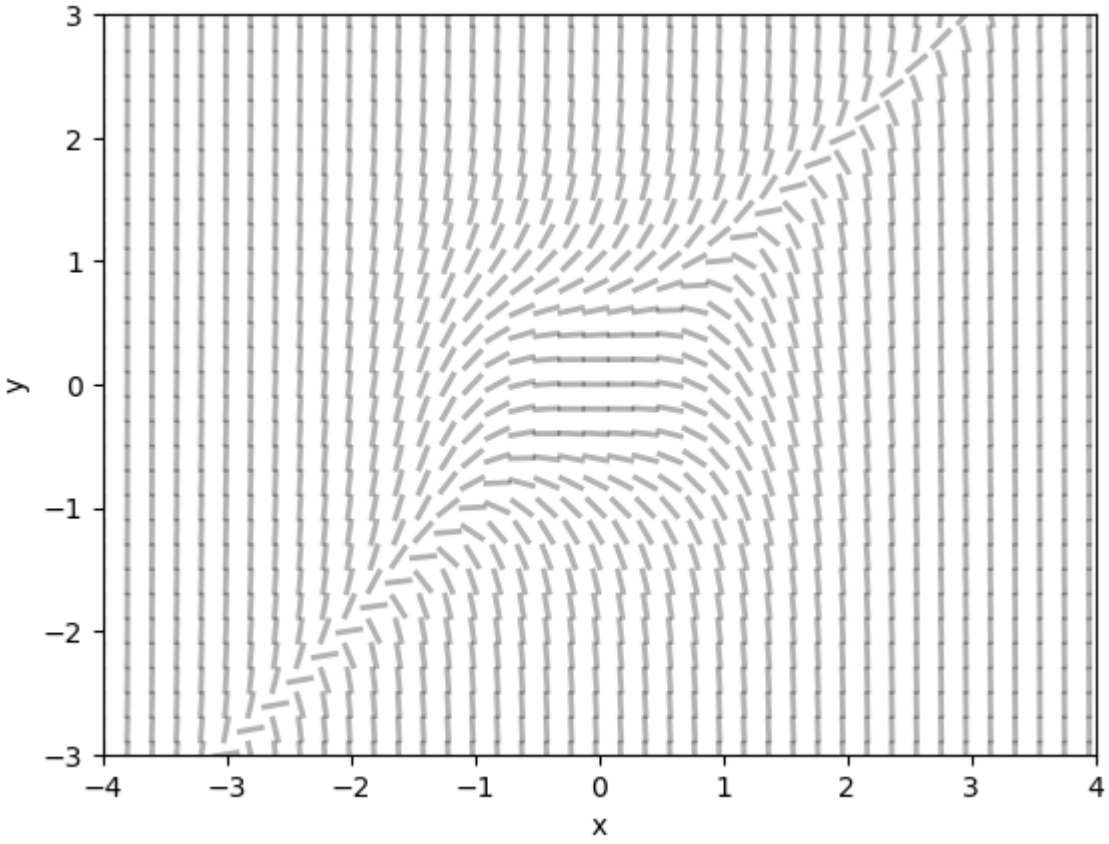
In [40]:
```
t = np.linspace(-10,20,200)

plt.plot( t, np.exp(-t + 5) );

plt.grid()
```



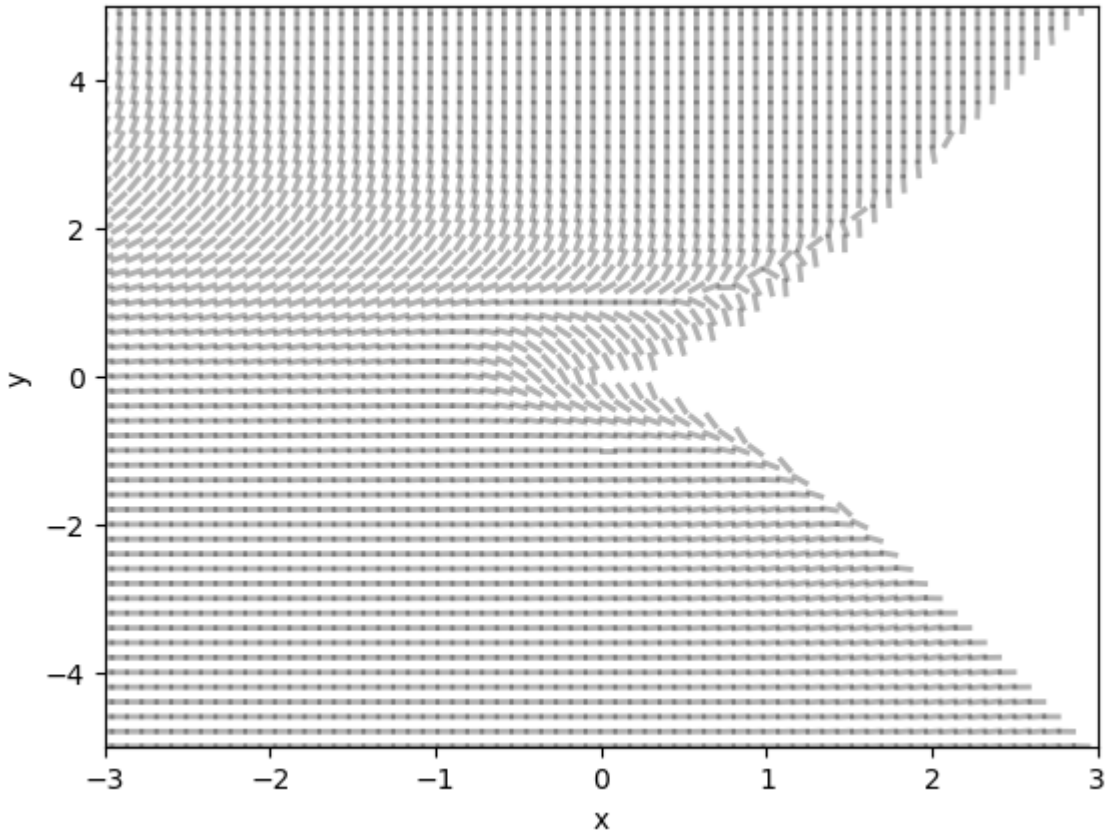
In [41]:
```
t = np.linspace(-15,15,200)

plt.plot( t, np.log(t**2) );

plt.grid()
```



In [42]:
```
from resources306 import *
def f(x,y): return y**3 - x**3
slopefieldplot( f, -4,4, -3,3, .2 ,lw=2)
plt.xlabel('x')
plt.ylabel('y');
```



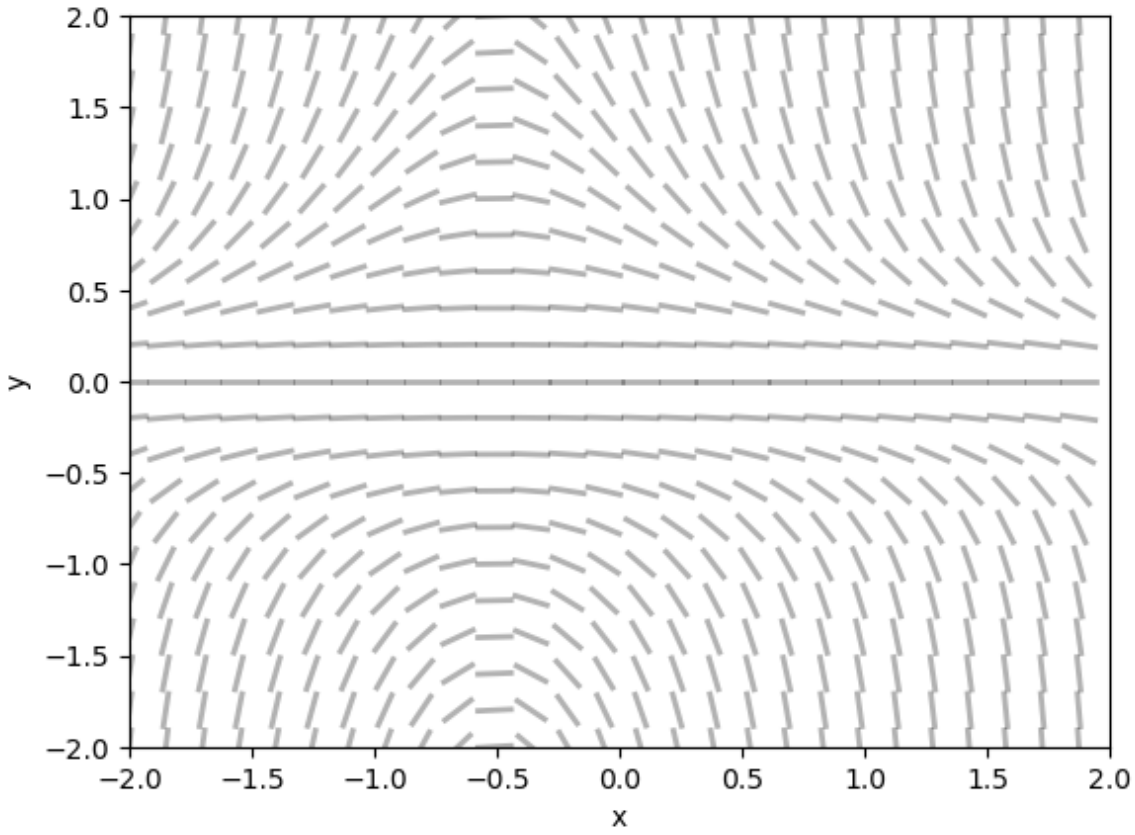
In [43]:
```
from resources306 import *
def f(x,y): return np.exp(y + x) * np.log(y**2 - x**3)
slopefieldplot( f, -3,3, -5,5, .2 ,lw=2)
plt.xlabel('x')
plt.ylabel('y');
```



In [44]:
```
# y = 1/x^2 + 1/x - 1/c
```

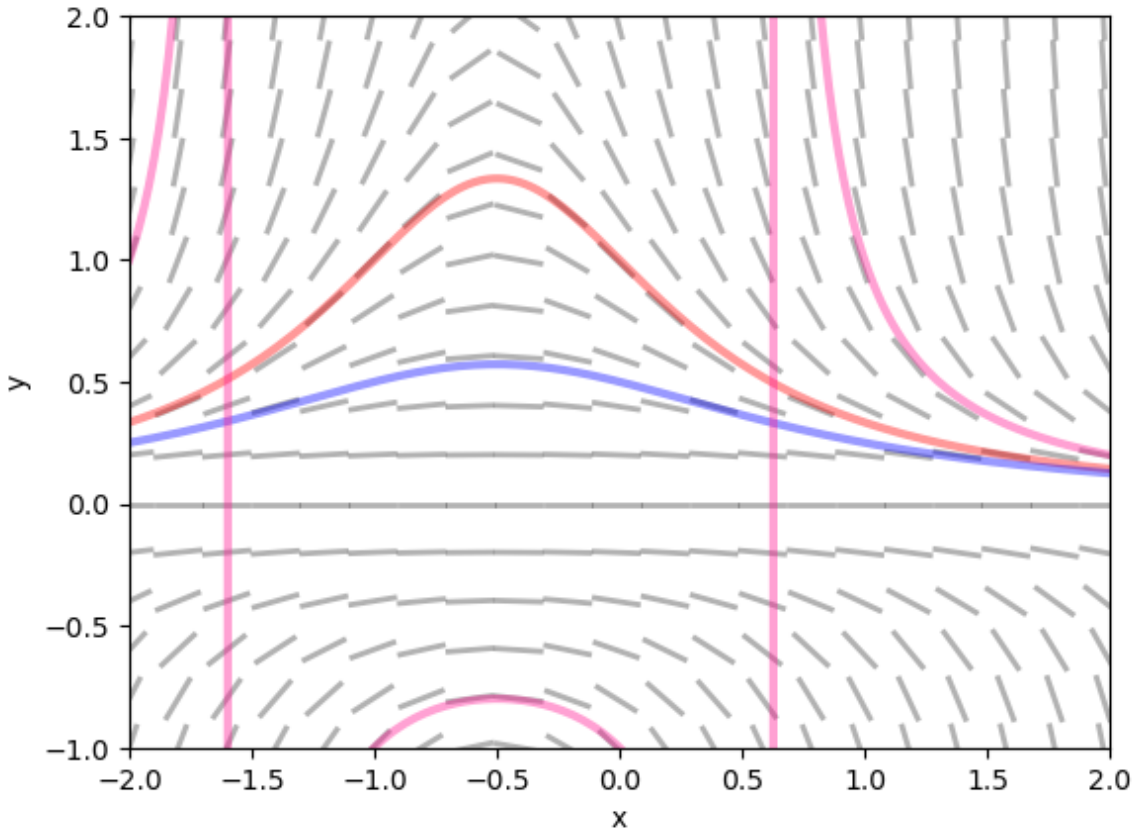
In [45]:
```
# c = -1 - 2 - 1
```

In [46]:
```
from resources306 import *
def f(x,y): return -2*x*y**2 - y**2
slopefieldplot( f, -2,2, -2,2, .2 ,lw=2)
plt.xlabel('x')
plt.ylabel('y');
```



In [47]:
```
x = sp.symbols('x')
y1 = 1/(x**2 + x + 1)
y2 = 1/(x**2 + x + 2)
y3 = 1/(x**2 + x - 1)

slopefieldplot( f, -2,2, -1,2, .2 ,lw=2)

expressionplot(y1,x,-2,2,color='r',alpha=.4,lw=3)
expressionplot(y2,x,-2,2,color='b',alpha=.4,lw=3)
expressionplot(y3,x,-2,2,color='deeppink',alpha=.4,lw=3)

plt.xlabel('x')
plt.ylabel('y')
```

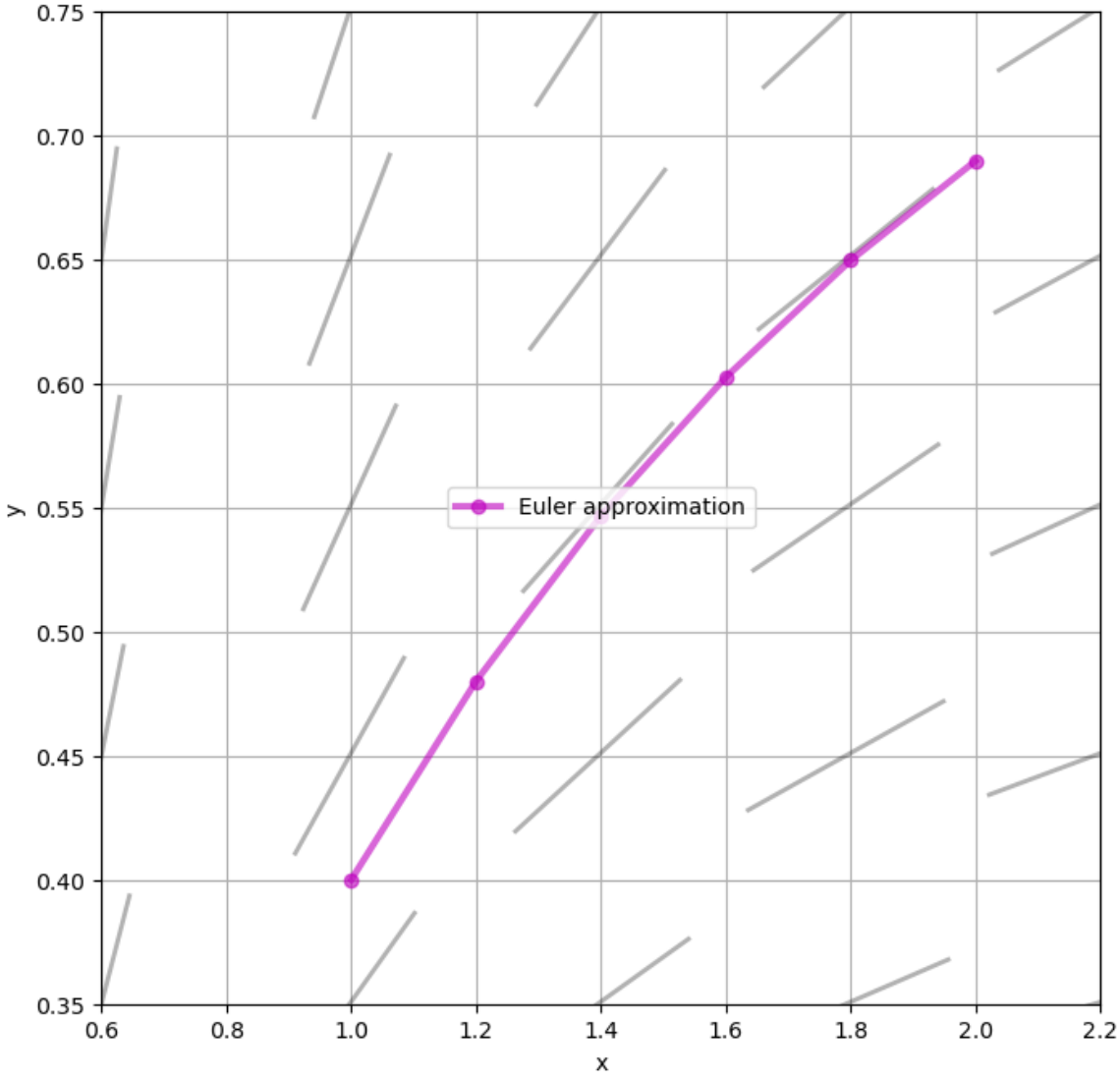
Out[47]:
```
Text(0, 0.5, 'y')
```



In [48]:
```
from resources306 import *
def f(x,y):
    return y/x**2

plt.figure(figsize=(8,8))
slopefieldplot( f, 0.6,2.2, 0.35,0.75, .1 ,lw=2)
y = 0.4 # This is the initial value of y.
x = 1 # This is the initial time.
xfinal = 2 # This is the value of x we want to get to.
n = 5 # Here we say how many steps we want to take.
h =(xfinal-x)/n # This is our step-size.
xlist = [x] # Initialize lists to store the data in
ylist = [y] # for later plotting.

for i in range(n): # Take n steps
    slope = f(x,y) # Compute the slope at the current location with DE
    y = y + h*slope # Take the Euler step to the new value of y.
    x = x + h # Advance x by one step.
    xlist.append(x) # Tack the new values at the ends of the lists.
    ylist.append(y)

for x,y in zip(xlist,ylist):
    print(f'{x:8.4f} {y:12.8f}')


plt.plot( xlist, ylist, 'mo-', lw=3, alpha=0.6, label='Euler approxima
plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend(loc='center')
```

```
  1.0000   0.40000000
  1.2000   0.48000000
  1.4000   0.54666667
  1.6000   0.60244898
  1.8000   0.64951531
  2.0000   0.68960884
```

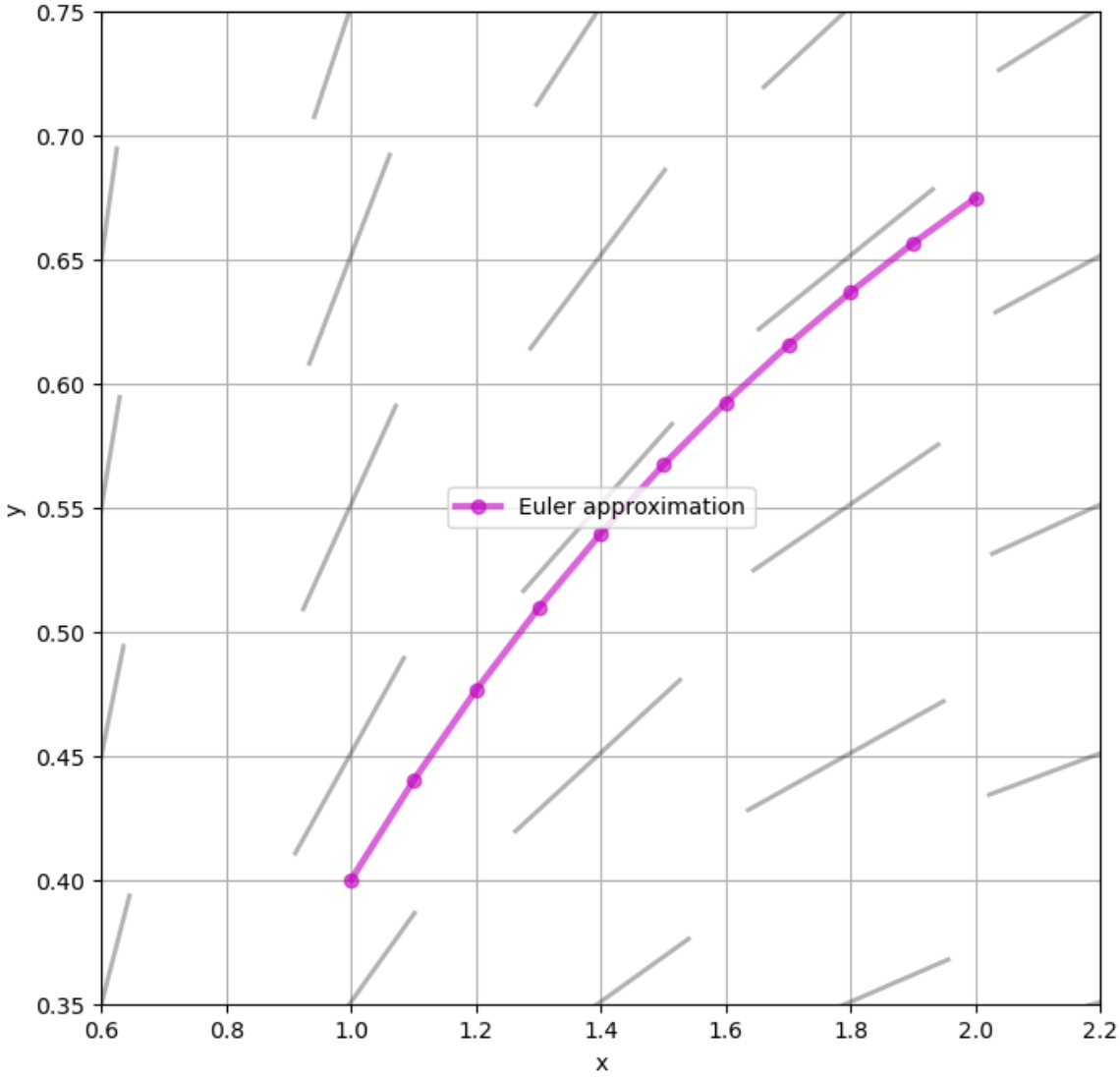
Out[48]:
```
<matplotlib.legend.Legend at 0x7f88f994ecd0>
```


0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
y
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
x
Euler approximation

In [49]:
```
from resources306 import *
def f(x,y):
    return y/x**2

plt.figure(figsize=(8,8))
slopefieldplot( f, 0.6,2.2, 0.35,0.75, .1 ,lw=2)
y = 0.4 # This is the initial value of y.
x = 1 # This is the initial time.
xfinal = 2 # This is the value of x we want to get to.
n = 10 # Here we say how many steps we want to take.
h =(xfinal-x)/n # This is our step-size.
xlist = [x] # Initialize lists to store the data in
ylist = [y] # for later plotting.

for i in range(n): # Take n steps
    slope = f(x,y) # Compute the slope at the current location with DE
    y = y + h*slope # Take the Euler step to the new value of y.
    x = x + h # Advance x by one step.
    xlist.append(x) # Tack the new values at the ends of the lists.
    ylist.append(y)

for x,y in zip(xlist,ylist):
    print(f'{x:8.4f} {y:12.8f}')


plt.plot( xlist, ylist, 'mo-', lw=3, alpha=0.6, label='Euler approxima
plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend(loc='center')
```

```
  1.0000   0.40000000
  1.1000   0.44000000
  1.2000   0.47636364
  1.3000   0.50944444
  1.4000   0.53958909
  1.5000   0.56711914
  1.6000   0.59232444
  1.7000   0.61546211
  1.8000   0.63675838
  1.9000   0.65641141
  2.0000   0.67459455
```

Out[49]: <matplotlib.legend.Legend at 0x7f88f854bc70>

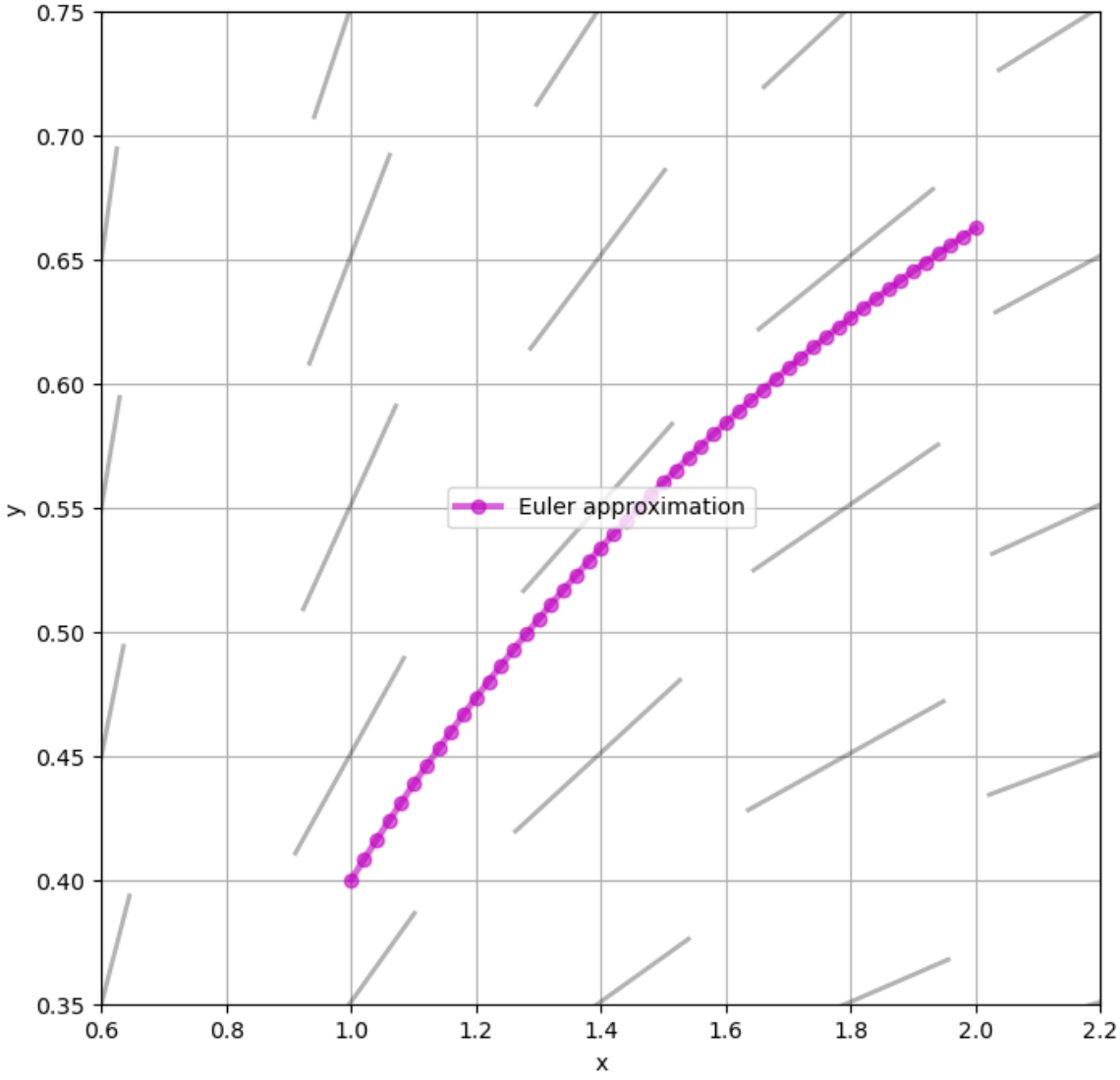

0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
y
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
x
Euler approximation

In [50]:
```
from resources306 import *
def f(x,y):
    return y/x**2

plt.figure(figsize=(8,8))
slopefieldplot( f, 0.6,2.2, 0.35,0.75, .1 ,lw=2)
y = 0.4 # This is the initial value of y.
x = 1 # This is the initial time.
xfinal = 2 # This is the value of x we want to get to.
n = 50 # Here we say how many steps we want to take.
h =(xfinal-x)/n # This is our step-size.
xlist = [x] # Initialize lists to store the data in
ylist = [y] # for later plotting.

for i in range(n): # Take n steps
    slope = f(x,y) # Compute the slope at the current location with DE
    y = y + h*slope # Take the Euler step to the new value of y.
    x = x + h # Advance x by one step.
    xlist.append(x) # Tack the new values at the ends of the lists.
    ylist.append(y)

for x,y in zip(xlist,ylist):
    print(f'{x:8.4f} {y:12.8f}')


plt.plot( xlist, ylist, 'mo-', lw=3, alpha=0.6, label='Euler approxima
plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend(loc='center')
```

```
  1.0000   0.40000000
  1.0200   0.40800000
  1.0400   0.41584314
  1.0600   0.42353254
  1.0800   0.43107139
  1.1000   0.43846288
  1.1200   0.44571020
  1.1400   0.45281655
  1.1600   0.45978510
  1.1800   0.46661900
  1.2000   0.47332137
  1.2200   0.47989528
  1.2400   0.48634375
  1.2600   0.49266977
  1.2800   0.49887624
  1.3000   0.50496604
  1.3200   0.51094197
  1.3400   0.51680677
  1.3600   0.52256314
  1.3800   0.52821369
  1.4000   0.53376099
  1.4200   0.53920753
  1.4400   0.54455575
  1.4600   0.54980803
```

```
1.4800   0.55496667
1.5000   0.56003393
1.5200   0.56501201
1.5400   0.56990304
1.5600   0.57470911
1.5800   0.57943222
1.6000   0.58407437
1.6200   0.58863745
1.6400   0.59312333
1.6600   0.59753383
1.6800   0.60187069
1.7000   0.60613565
1.7200   0.61033036
1.7400   0.61445645
1.7600   0.61851548
1.7800   0.62250899
1.8000   0.62643848
1.8200   0.63030538
1.8400   0.63411111
1.8600   0.63785704
1.8800   0.64154451
1.9000   0.64517479
1.9200   0.64874917
1.9400   0.65226886
1.9600   0.65573506
1.9800   0.65914892
2.0000   0.66251158
```

Out[50]: <matplotlib.legend.Legend at 0x7f88eba7b970>


0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
y
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
x
Euler approximation

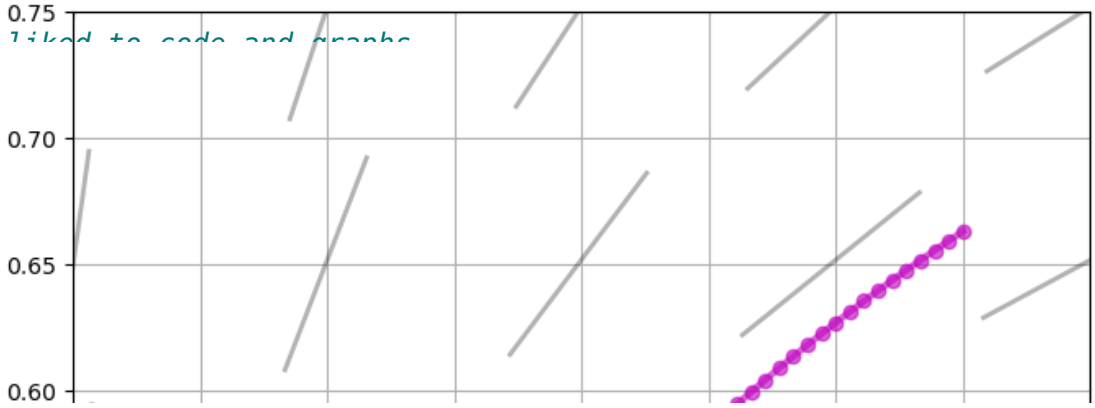
In [51]:
```
from resources306 import *
def f(x,y):
    return y/x**2

plt.figure(figsize=(8,8))
slopefieldplot( f, 0.6,2.2, 0.35,0.75, .1 ,lw=2)
y = 0.4 # This is the initial value of y.
x = 1 # This is the initial time.
xfinal = 2 # This is the value of x we want to get to.
n = 50 - 5 # Here we say how many steps we want to take.
h =(xfinal-x)/n # This is our step-size.
xlist = [x] # Initialize lists to store the data in
ylist = [y] # for later plotting.

for i in range(n): # Take n steps
    slope = f(x,y) # Compute the slope at the current location with DE
    y = y + h*slope # Take the Euler step to the new value of y.
    x = x + h # Advance x by one step.
    xlist.append(x) # Tack the new values at the ends of the lists.
    ylist.append(y)

for x,y in zip(xlist,ylist):
    print(f'{x:8.4f} {y:12.8f}')


plt.plot( xlist, ylist, 'mo-', lw=3, alpha=0.6, label='Euler approxima
plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend(loc='center')
```

```
  1.0000   0.40000000
  1.0222   0.40888889
  1.0444   0.41758454
  1.0667   0.42609124
  1.0889   0.43441334
  1.1111   0.44255520
  1.1333   0.45052119
  1.1556   0.45831567
  1.1778   0.46594297
  1.2000   0.47340734
  1.2222   0.48071301
  1.2444   0.48786412
  1.2667   0.49486472
  1.2889   0.50171880
  1.3111   0.50843026
  1.3333   0.51500290
  1.3556   0.52144043
  1.3778   0.52774649
  1.4000   0.53392458
  1.4222   0.53997815
  1.4444   0.54591052
  1.4667   0.55172495
  1.4889   0.55742459
  1.5111   0.56301250
```

```
1.5333   0.56849164
1.5556   0.57386491
1.5778   0.57913510
1.6000   0.58430492
1.6222   0.58937701
1.6444   0.59435392
1.6667   0.59923813
1.6889   0.60403204
1.7111   0.60873797
1.7333   0.61335817
1.7556   0.61789485
1.7778   0.62235010
1.8000   0.62672600
1.8222   0.63102454
1.8444   0.63524763
1.8667   0.63939717
1.8889   0.64347496
1.9111   0.64748276
1.9333   0.65142228
1.9556   0.65529519
1.9778   0.65910307
2.0000   0.66284750
```

Out[51]: <matplotlib.legend.Legend at 0x7f88ebee6310>

In [52]: #I liked to code and graphs



In [ ]: