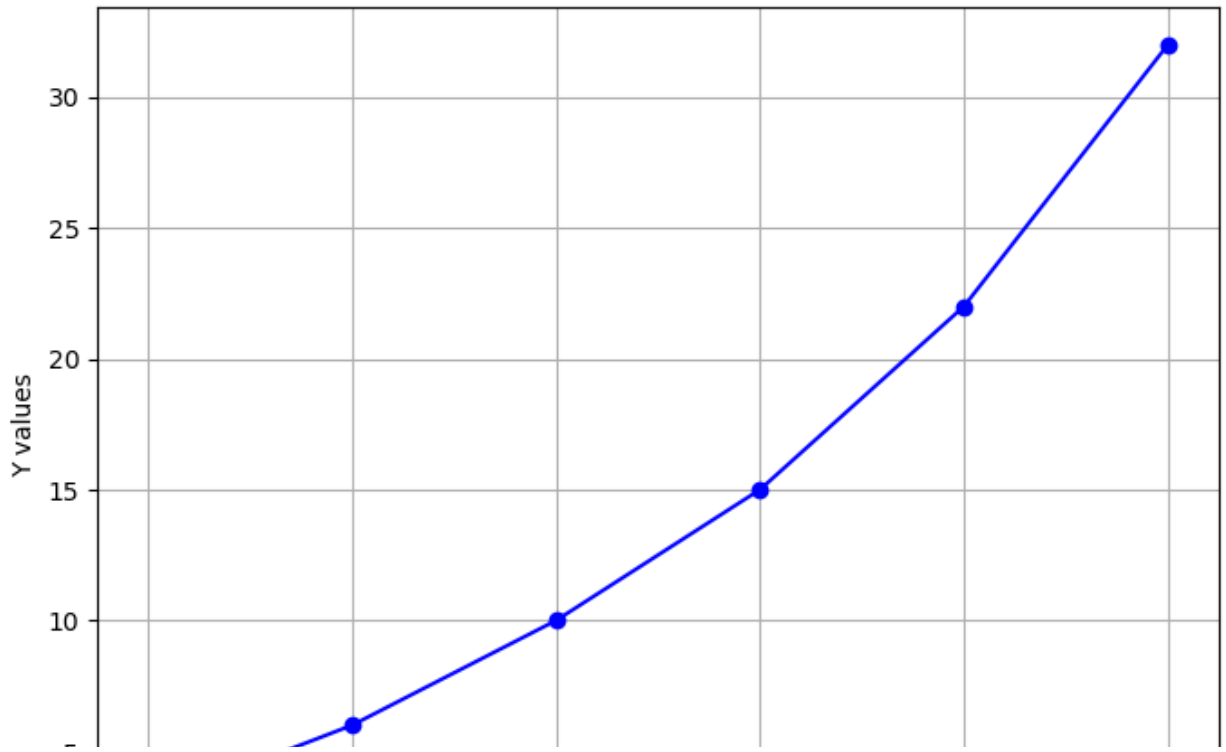
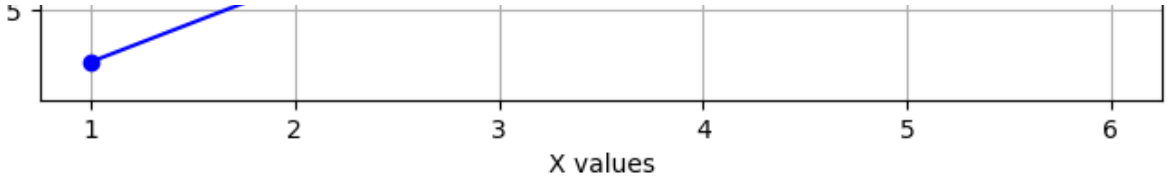
In [96]:
```
### Training with fancier version ###

import torch
import torch.nn as nn
import torch.nn.functional as F

import torch.optim as optim
import matplotlib.pyplot as plt

class Net(nn.Module): ## nn.Module class is used
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1,4096,bias=False)  # in dim, out dim
        self.fc2 = nn.Linear(4096,2048,bias=False)
        self.fc3 = nn.Linear(2048,1,bias=False)

    def forward(self, x):
        x = self.fc1(x)
        x = F.sigmoid(x)
        x = self.fc2(x)
        x = F.sigmoid(x)
        x = self.fc3(x)
        return x

net = Net()

print(net)
print(list(net.parameters())) # parameters are randomized


#def criterion(out, label):
#    return (label - out)**2
criterion = nn.MSELoss()


# optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam(net.parameters(), lr=0.00007,)


data = [(1.0,3.0), (2.0,6.0), (3.0,10.0), (4.0,15.0), (5.0,22.0), (6.0

# Split data into x and y
x_values, y_values = zip(*data)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, marker='o', linestyle='-', color='b')
plt.xlabel('X values')
```

```
plt.ylabel('Y values')
plt.grid(True)
plt.show()
```

```
Net(
  (fc1): Linear(in_features=1, out_features=4096, bias=False)
  (fc2): Linear(in_features=4096, out_features=2048, bias=False)
  (fc3): Linear(in_features=2048, out_features=1, bias=False)
)
[Parameter containing:
tensor([[ 0.2647],
        [ 0.6125],
        [-0.9204],
        ...,
        [ 0.0500],
        [-0.9578],
        [ 0.2392]], requires_grad=True), Parameter containing:
tensor([[-0.0024, -0.0130, -0.0089,  ...,  0.0104, -0.0093,  0.0004],
        [-0.0113,  0.0020, -0.0090,  ..., -0.0022,  0.0090,  0.0082],
        [ 0.0095, -0.0065,  0.0018,  ..., -0.0002, -0.0011, -0.0053],
        ...,
        [ 0.0075, -0.0020, -0.0113,  ...,  0.0152, -0.0037, -0.0074],
        [ 0.0107,  0.0123,  0.0095,  ...,  0.0040, -0.0150,  0.0105],
        [ 0.0084,  0.0105,  0.0039,  ...,  0.0059, -0.0142, -0.001
0]],
       requires_grad=True), Parameter containing:
tensor([[ 0.0178,  0.0086,  0.0049,  ..., -0.0132, -0.0218, -0.017
1]],
       requires_grad=True)]
```





In [97]:
```
for epoch in range(1500): # 0 - 19
    for i, current_data in enumerate(data):
        X, Y = current_data
        X, Y = torch.FloatTensor([X]), torch.FloatTensor([Y])
        optimizer.zero_grad()
        outputs = net(X)
        loss = criterion(outputs, Y)
        loss.backward()
        optimizer.step()    ## This line is equivalent to "W = W - lr*
    print("Epoch {} - loss: {}".format(epoch, loss))

### Test the trained network ###
for i, current_data in enumerate(data):
    X, Y = current_data
    X, Y = torch.FloatTensor([X]), torch.FloatTensor([Y])
    out = net(torch.FloatTensor(X))
    print("when x = {}, y = {}".format(X, out))
```

```
Epoch 1486 - loss: 1.4406759873963892e-07
Epoch 1487 - loss: 1.0762596502900124e-07
Epoch 1488 - loss: 1.0637813829816878e-07
Epoch 1489 - loss: 7.754715625196695e-08
Epoch 1490 - loss: 7.968628779053688e-08
Epoch 1491 - loss: 5.6843418860808015e-08
Epoch 1492 - loss: 5.6843418860808015e-08
Epoch 1493 - loss: 4.165121936239302e-08
Epoch 1494 - loss: 4.165121936239302e-08
Epoch 1495 - loss: 2.8816430130973458e-08
Epoch 1496 - loss: 3.0126102501526475e-08
Epoch 1497 - loss: 2.0463630789890885e-08
Epoch 1498 - loss: 2.386877895332873e-08
Epoch 1499 - loss: 1.1819793144240975e-08
when x = tensor([1.]), y = tensor([3.0000], grad_fn=<SqueezeBackward4
>)
when x = tensor([2.]), y = tensor([6.0000], grad_fn=<SqueezeBackward4
>)
when x = tensor([3.]), y = tensor([10.0000], grad_fn=<SqueezeBackward
4>)
```

In [ ]:
```
for epoch in range(20):
    print(epoch)
```

In [ ]:
```
W = torch.tensor([1.0], requires_grad=True)
W = W*2
label = 1.0
loss = W*5 - label
loss.backward()
W.grad
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: