Gradient Descent

In [1]:

```
#IMPORT
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import sklearn.datasets as dt
from sklearn.model_selection import train_test_split

## Set a seed for the random number generator
np.random.seed(100)
```

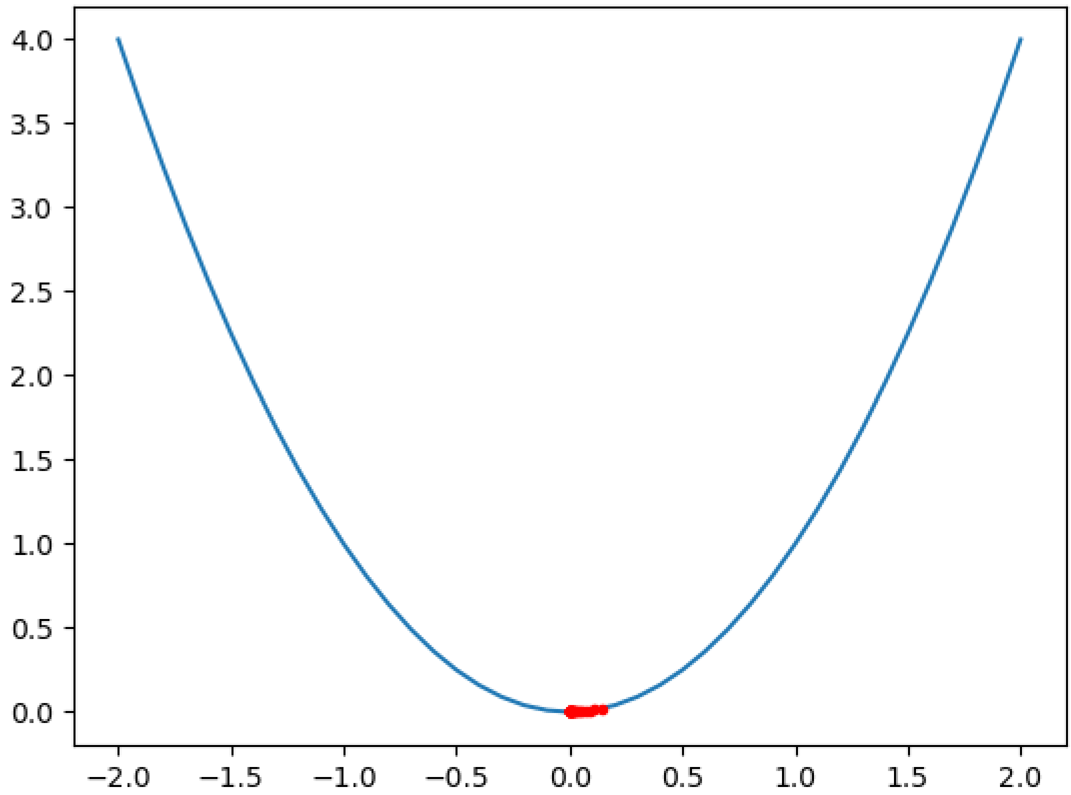
Gradient descent algorithm - basic version

In [2]:

```
# example of plotting a gradient descent search on a single-variable function
from numpy import asarray
from numpy import arange
from numpy.random import rand
from matplotlib import pyplot

# objective function
def objective(x):
        return x**2

# derivative of objective function
def derivative(x):
        return 2 * x

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
        # track all solutions
        solutions, scores = list(), list()
        # generate an initial point
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
        # run the gradient descent
        for i in range(n_iter):
                # calculate gradient
                gradient = derivative(solution)
                # take a step
                solution = solution - step_size * gradient
                # evaluate candidate point
                solution_eval = objective(solution)
                # store solution
                solutions.append(solution)
                scores.append(solution_eval)
                # report progress
                print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
        return [solutions, scores]

# define range for input
bounds = asarray([[-2.0, 2]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1
# perform the gradient descent search
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_si
# sample input range uniformly at 0.1 increments
```

```
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

```
>0 f([0.13889581]) = 0.01929
>1 f([0.11111665]) = 0.01235
>2 f([0.08889332]) = 0.00790
>3 f([0.07111466]) = 0.00506
>4 f([0.05689173]) = 0.00324
>5 f([0.04551338]) = 0.00207
>6 f([0.0364107]) = 0.00133
>7 f([0.02912856]) = 0.00085
>8 f([0.02330285]) = 0.00054
>9 f([0.01864228]) = 0.00035
>10 f([0.01491382]) = 0.00022
>11 f([0.01193106]) = 0.00014
>12 f([0.00954485]) = 0.00009
>13 f([0.00763588]) = 0.00006
>14 f([0.0061087]) = 0.00004
>15 f([0.00488696]) = 0.00002
>16 f([0.00390957]) = 0.00002
>17 f([0.00312766]) = 0.00001
>18 f([0.00250212]) = 0.00001
>19 f([0.0020017]) = 0.00000
>20 f([0.00160136]) = 0.00000
>21 f([0.00128109]) = 0.00000
>22 f([0.00102487]) = 0.00000
>23 f([0.0008199]) = 0.00000
>24 f([0.00065592]) = 0.00000
>25 f([0.00052473]) = 0.00000
>26 f([0.00041979]) = 0.00000
>27 f([0.00033583]) = 0.00000
>28 f([0.00026866]) = 0.00000
>29 f([0.00021493]) = 0.00000
```



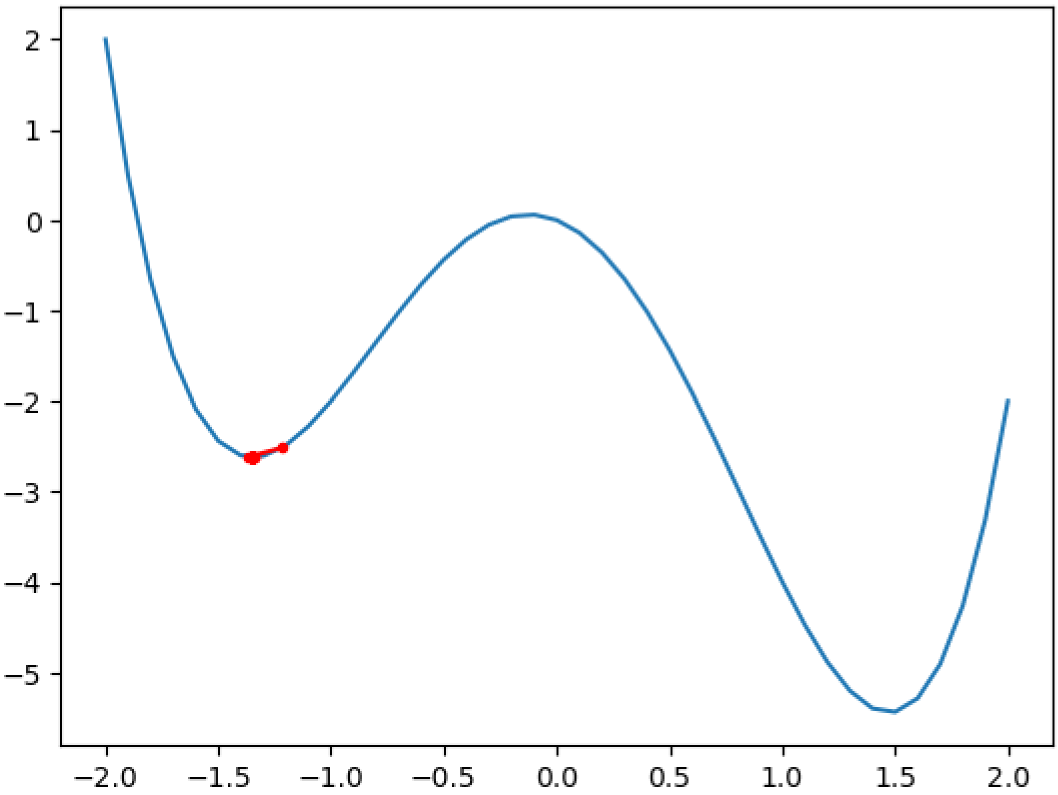
Task 1 - adapt the basic algorithm to use a variable step-size γ_i. Choose some different formulas for γ_i and compare the performance (ie. rate of convergence) between them and with a fixed step-size. You may want to test with other objective functions as well.

Gradient descent - local vs. global minima

In [3]:
```
# example of plotting a gradient descent search on a single-variable function
# objective function
def objective(x):
        return x**4-4*x**2-x

# derivative of objective function
def derivative(x):
        return 4*x ** 3-8*x-1

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
        # track all solutions
        solutions, scores = list(), list()
        # generate an initial point
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
        # run the gradient descent
        for i in range(n_iter):
                # calculate gradient
                gradient = derivative(solution)
                # take a step
                solution = solution - step_size * gradient
                # evaluate candidate point
                solution_eval = objective(solution)
                # store solution
                solutions.append(solution)
                scores.append(solution_eval)
                # report progress
                print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
        return [solutions, scores]
```

```
# define range for input
bounds = asarray([[-2.0, 2.0]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1
# perform the gradient descent search
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_si
# sample input range uniformly at 0.1 increments
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

```
>0 f([-1.2170454]) = -2.51380
>1 f([-1.3696069]) = -2.61497
>2 f([-1.33763634]) = -2.61796
>3 f([-1.35038786]) = -2.61848
>4 f([-1.34569965]) = -2.61854
>5 f([-1.34748431]) = -2.61855
>6 f([-1.34681333]) = -2.61856
>7 f([-1.3470668]) = -2.61856
>8 f([-1.34697122]) = -2.61856
>9 f([-1.34700729]) = -2.61856
>10 f([-1.34699368]) = -2.61856
>11 f([-1.34699881]) = -2.61856
>12 f([-1.34699688]) = -2.61856
>13 f([-1.34699761]) = -2.61856
>14 f([-1.34699733]) = -2.61856
>15 f([-1.34699744]) = -2.61856
>16 f([-1.3469974]) = -2.61856
>17 f([-1.34699741]) = -2.61856
>18 f([-1.34699741]) = -2.61856
>19 f([-1.34699741]) = -2.61856
>20 f([-1.34699741]) = -2.61856
>21 f([-1.34699741]) = -2.61856
>22 f([-1.34699741]) = -2.61856
>23 f([-1.34699741]) = -2.61856
>24 f([-1.34699741]) = -2.61856
>25 f([-1.34699741]) = -2.61856
>26 f([-1.34699741]) = -2.61856
>27 f([-1.34699741]) = -2.61856
>28 f([-1.34699741]) = -2.61856
>29 f([-1.34699741]) = -2.61856
```



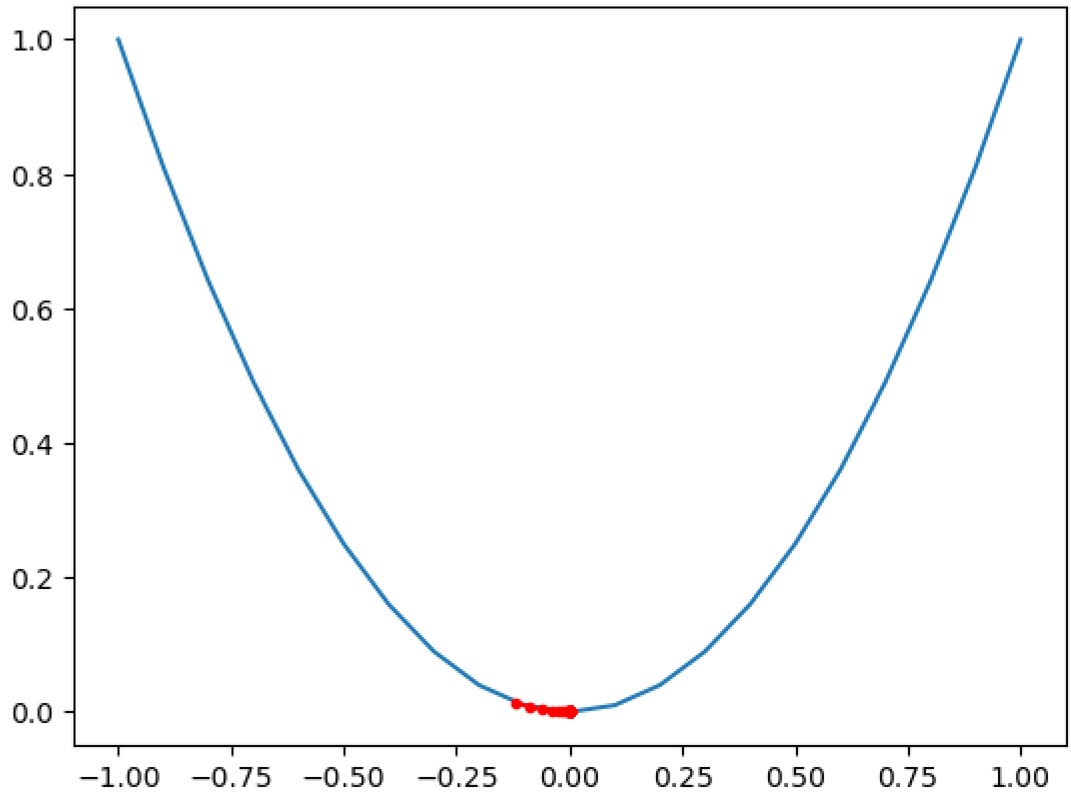
Gradient descent with momentum

Momentum - weight by previous points, allows the search to build inertia in a direction in the search space, can overcome the oscillations of noisy gradients and navigate flat spots

In [4]:
```
# example of plotting gradient descent with momentum for a single-variable functior
# objective function
def objective(x):
        return x**2.0

# derivative of objective function
def derivative(x):
        return x * 2.0

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size, momentum):
        # track all solutions
        solutions, scores = list(), list()
        # generate an initial point
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
        # keep track of the change
        change = 0.0
        # run the gradient descent
        for i in range(n_iter):
                # calculate gradient
                gradient = derivative(solution)
                # calculate update
                new_change = step_size * gradient + momentum * change
                # take a step
                solution = solution - new_change
                # save the change
                change = new_change
                # evaluate candidate point
                solution_eval = objective(solution)
                # store solution
```

```
                solutions.append(solution)
                scores.append(solution_eval)
                # report progress
                print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
        return [solutions, scores]


# define range for input
bounds = asarray([[-1.0, 1.0]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1
# define momentum
momentum = 0.3
# perform the gradient descent search with momentum
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_si
# sample input range uniformly at 0.1 increments
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

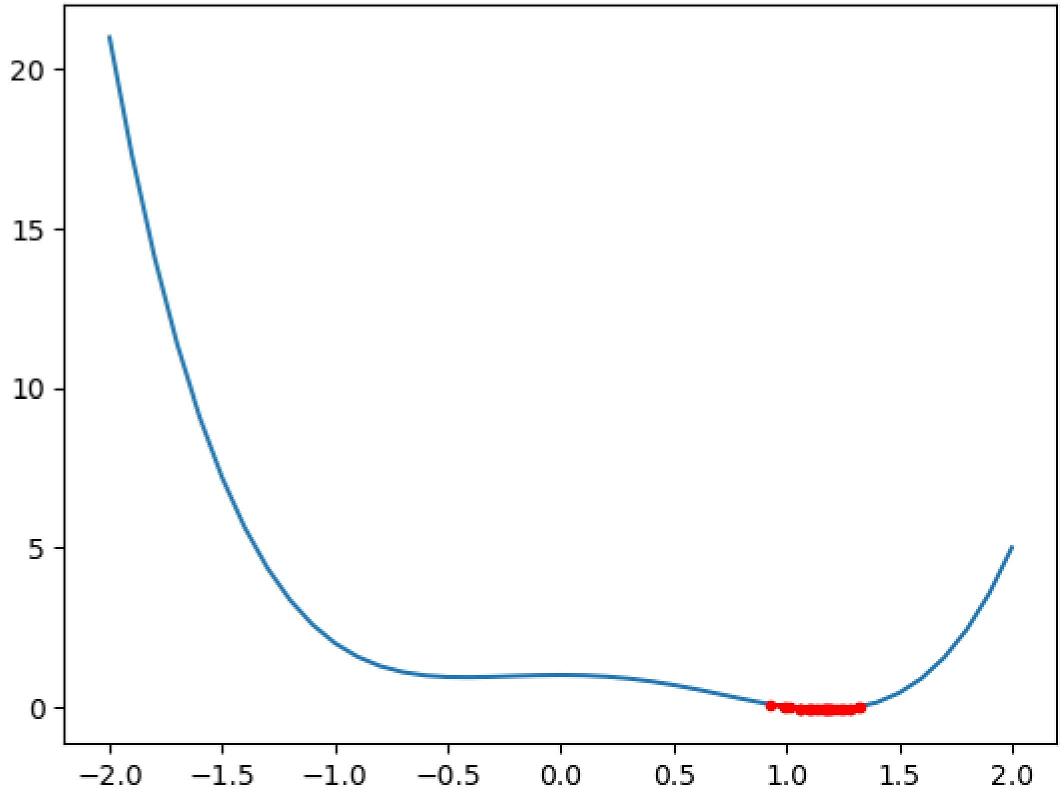
```
>0 f([-0.12077185]) = 0.01459
>1 f([-0.08755959]) = 0.00767
>2 f([-0.060084]) = 0.00361
>3 f([-0.03982452]) = 0.00159
>4 f([-0.02578177]) = 0.00066
>5 f([-0.01641259]) = 0.00027
>6 f([-0.01031932]) = 0.00011
>7 f([-0.00642748]) = 0.00004
>8 f([-0.00397443]) = 0.00002
>9 f([-0.00244363]) = 0.00001
>10 f([-0.00149566]) = 0.00000
>11 f([-0.00091214]) = 0.00000
>12 f([-0.00055465]) = 0.00000
>13 f([-0.00033648]) = 0.00000
>14 f([-0.00020373]) = 0.00000
>15 f([-0.00012316]) = 0.00000
>16 f([-7.43563618e-05]) = 0.00000
>17 f([-4.48441711e-05]) = 0.00000
>18 f([-2.70216797e-05]) = 0.00000
>19 f([-1.62705963e-05]) = 0.00000
>20 f([-9.79115205e-06]) = 0.00000
>21 f([-5.88908835e-06]) = 0.00000
>22 f([-3.54065158e-06]) = 0.00000
>23 f([-2.12799023e-06]) = 0.00000
>24 f([-1.27859378e-06]) = 0.00000
>25 f([-7.68056087e-07]) = 0.00000
>26 f([-4.61283562e-07]) = 0.00000
>27 f([-2.76995093e-07]) = 0.00000
>28 f([-1.66309533e-07]) = 0.00000
>29 f([-9.98419586e-08]) = 0.00000
```



In [6]:
```
# example of plotting gradient descent with momentum for a single-variable function
# objective function
def objective(x):
        return x**4-x**3-x**2+1

# derivative of objective function
def derivative(x):
        return 4*x ** 3-3*x**2-2*x

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size, momentum):
        # track all solutions
        solutions, scores = list(), list()
        # generate an initial point
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
        # keep track of the change
        change = 0.0
        # run the gradient descent
        for i in range(n_iter):
                # calculate gradient
                gradient = derivative(solution)
                # calculate update
                new_change = step_size * gradient + momentum * change
                # take a step
                solution = solution - new_change
                # save the change
                change = new_change
                # evaluate candidate point
                solution_eval = objective(solution)
                # store solution
                solutions.append(solution)
                scores.append(solution_eval)
                # report progress
                print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
        return [solutions, scores]
```

```
# define range for input
bounds = asarray([[-2.0, 2.0]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1
# define momentum
momentum = 0.9
# perform the gradient descent search with momentum
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_si
# sample input range uniformly at 0.1 increments
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

```
>0 f([1.1763205]) = -0.09673
>1 f([0.993114]) = 0.00698
>2 f([0.93094001]) = 0.07763
>3 f([0.9984468]) = 0.00156
>4 f([1.15982203]) = -0.09584
>5 f([1.31650921]) = -0.01100
>6 f([1.3280808]) = 0.00472
>7 f([1.19626415]) = -0.09506
>8 f([1.06143184]) = -0.05317
>9 f([1.01202081]) = -0.01173
>10 f([1.06261181]) = -0.05402
>11 f([1.17947259]) = -0.09667
>12 f([1.28155653]) = -0.04976
>13 f([1.28053474]) = -0.05071
>14 f([1.18774026]) = -0.09615
>15 f([1.09476049]) = -0.07417
>16 f([1.06475254]) = -0.05553
>17 f([1.10796217]) = -0.08074
>18 f([1.19267237]) = -0.09559
>19 f([1.25557121]) = -0.07059
>20 f([1.24448949]) = -0.07752
>21 f([1.17707679]) = -0.09672
>22 f([1.11513309]) = -0.08387
>23 f([1.10078999]) = -0.07730
>24 f([1.1380129]) = -0.09167
>25 f([1.19811402]) = -0.09475
>26 f([1.23452485]) = -0.08280
>27 f([1.21882322]) = -0.08933
>28 f([1.169876]) = -0.09662
>29 f([1.12994013]) = -0.08930
```



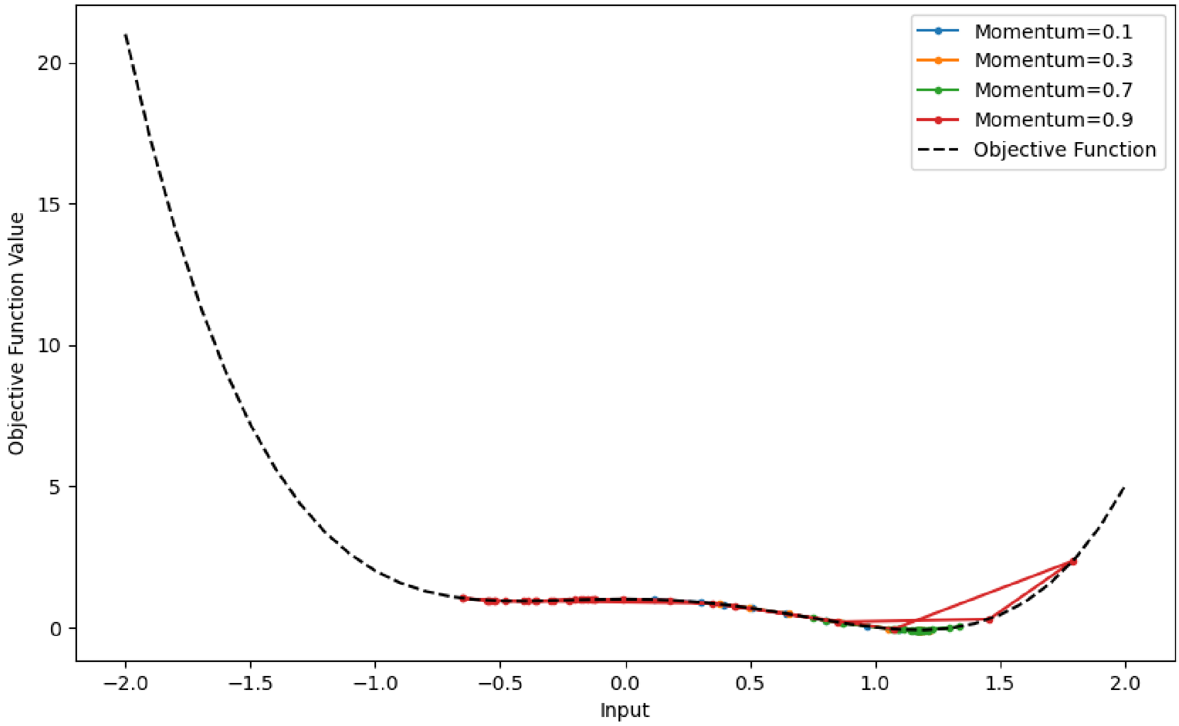
Task 2 - Use momentum values of 0.1, 0.3, 0.7, 0.9 in the code above, with objective function $f(x) = x^4 - x^{-}x^2 + 1$, and compare your results. Explain any differences that you see, especially whether there is convergence to the global minimum.

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt

# objective function
def objective(x):
    return x**4 - x**3 - x**2 + 1

# derivative of objective function
def derivative(x):
    return 4*x**3 - 3*x**2 - 2*x

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size, momentum):
    # track all solutions
    solutions, scores = list(), list()
    # generate an initial point
    solution = bounds[:, 0] + np.random.rand(len(bounds)) * (bounds[:, 1] - bounds[
    # keep track of the change
    change = 0.0
    # run the gradient descent
    for i in range(n_iter):
        # calculate gradient
        gradient = derivative(solution)
        # calculate update
        new_change = step_size * gradient + momentum * change
        # take a step
        solution = solution - new_change
        # save the change
        change = new_change
        # evaluate candidate point
        solution_eval = objective(solution)
        # store solution
```

```
        solutions.append(solution)
        scores.append(solution_eval)
        # report progress
        print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
    return [solutions, scores]

# define range for input
bounds = np.asarray([[-2.0, 2.0]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1

# perform the gradient descent search with different momentum values
momentum_values = [0.1, 0.3, 0.7, 0.9]

plt.figure(figsize=(10, 6))

for momentum in momentum_values:
    # perform gradient descent
    solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, ste
    # plot the solutions found
    plt.plot(solutions, scores, '.-', label=f'Momentum={momentum}')

# sample input range uniformly at 0.1 increments
inputs = np.arange(bounds[0, 0], bounds[0, 1] + 0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
plt.plot(inputs, results, label='Objective Function', linestyle='--', color='black'
# set plot labels and legend
plt.xlabel('Input')
plt.ylabel('Objective Function Value')
plt.legend()
# show the plot
plt.show()
```

```
>0 f([0.11718035]) = 0.98485
>1 f([0.30112758]) = 0.89024
>2 f([0.39602893]) = 0.80565
>3 f([0.50693143]) = 0.67879
>4 f([0.64439342]) = 0.48960
>5 f([0.80455925]) = 0.25090
>6 f([0.96736087]) = 0.03467
>7 f([1.09575181]) = -0.07470
>8 f([1.16168745]) = -0.09604
>9 f([1.1783867]) = -0.09670
>10 f([1.17779168]) = -0.09671
>11 f([1.17591862]) = -0.09673
>12 f([1.17533356]) = -0.09673
>13 f([1.17531792]) = -0.09673
>14 f([1.175371]) = -0.09673
>15 f([1.17539101]) = -0.09673
>16 f([1.17539265]) = -0.09673
>17 f([1.17539122]) = -0.09673
>18 f([1.17539056]) = -0.09673
>19 f([1.17539047]) = -0.09673
>20 f([1.17539051]) = -0.09673
>21 f([1.17539053]) = -0.09673
>22 f([1.17539053]) = -0.09673
>23 f([1.17539053]) = -0.09673
>24 f([1.17539053]) = -0.09673
>25 f([1.17539053]) = -0.09673
>26 f([1.17539053]) = -0.09673
>27 f([1.17539053]) = -0.09673
>28 f([1.17539053]) = -0.09673
>29 f([1.17539053]) = -0.09673
>0 f([0.37667485]) = 0.82480
>1 f([0.49608782]) = 0.69237
>2 f([0.65612471]) = 0.47237
>3 f([0.85152603]) = 0.18323
>4 f([1.05100516]) = -0.04540
>5 f([1.18805238]) = -0.09612
>6 f([1.21945818]) = -0.08911
>7 f([1.19352313]) = -0.09548
>8 f([1.17172821]) = -0.09668
>9 f([1.16793119]) = -0.09653
>10 f([1.17234448]) = -0.09670
>11 f([1.17595069]) = -0.09673
>12 f([1.17661062]) = -0.09673
>13 f([1.17588868]) = -0.09673
>14 f([1.17529691]) = -0.09673
>15 f([1.17518983]) = -0.09673
>16 f([1.17530871]) = -0.09673
>17 f([1.17540595]) = -0.09673
>18 f([1.17542351]) = -0.09673
>19 f([1.17540396]) = -0.09673
>20 f([1.17538798]) = -0.09673
>21 f([1.17538511]) = -0.09673
>22 f([1.17538833]) = -0.09673
>23 f([1.17539095]) = -0.09673
>24 f([1.17539142]) = -0.09673
>25 f([1.17539089]) = -0.09673
>26 f([1.17539046]) = -0.09673
>27 f([1.17539038]) = -0.09673
>28 f([1.17539047]) = -0.09673
>29 f([1.17539054]) = -0.09673
>0 f([1.07932225]) = -0.06520
>1 f([0.80155499]) = 0.25531
>2 f([0.75417948]) = 0.32576
>3 f([0.87090162]) = 0.15625
```

```
>4 f([1.09010736]) = -0.07161
>5 f([1.29990839]) = -0.03100
>6 f([1.33506511]) = 0.01493
>7 f([1.20956008]) = -0.09219
>8 f([1.09467748]) = -0.07412
>9 f([1.06798175]) = -0.05777
>10 f([1.11781701]) = -0.08496
>11 f([1.19242796]) = -0.09562
>12 f([1.23150864]) = -0.08422
>13 f([1.21306184]) = -0.09119
>14 f([1.17019972]) = -0.09663
>15 f([1.14407306]) = -0.09316
>16 f([1.14827762]) = -0.09404
>17 f([1.17081811]) = -0.09666
>18 f([1.19001455]) = -0.09592
>19 f([1.19220704]) = -0.09565
>20 f([1.18076945]) = -0.09663
>21 f([1.16868268]) = -0.09657
>22 f([1.16522054]) = -0.09635
>23 f([1.17033672]) = -0.09664
>24 f([1.17769332]) = -0.09671
>25 f([1.18110393]) = -0.09661
>26 f([1.17915503]) = -0.09668
>27 f([1.17494182]) = -0.09673
>28 f([1.17233005]) = -0.09670
>29 f([1.1727948]) = -0.09671
>0 f([-0.36039948]) = 0.93379
>1 f([0.34772439]) = 0.85166
>2 f([1.07403677]) = -0.06183
>3 f([1.79300757]) = 2.35626
>4 f([1.45742674]) = 0.29197
>5 f([0.84583335]) = 0.19127
>6 f([0.43714099]) = 0.76189
>7 f([0.18066004]) = 0.96253
>8 f([-0.00660795]) = 0.99996
>9 f([-0.17645752]) = 0.97533
>10 f([-0.35307469]) = 0.93489
>11 f([-0.52764061]) = 0.94600
>12 f([-0.64799764]) = 1.02851
>13 f([-0.65111028]) = 1.03182
>14 f([-0.54653647]) = 0.95377
>15 f([-0.40681605]) = 0.92922
>16 f([-0.28584999]) = 0.94832
>17 f([-0.20029473]) = 0.96953
>18 f([-0.14810437]) = 0.98179
>19 f([-0.12287399]) = 0.98699
>20 f([-0.11946998]) = 0.98764
>21 f([-0.13533636]) = 0.98450
>22 f([-0.17019707]) = 0.97680
>23 f([-0.22494897]) = 0.96334
>24 f([-0.29948171]) = 0.94522
>25 f([-0.38880661]) = 0.93046
>26 f([-0.47809872]) = 0.93295
>27 f([-0.54179463]) = 0.95166
>28 f([-0.55580179]) = 0.95821
>29 f([-0.51821555]) = 0.94274
```



Task 3 - Use the gradient descent with momentum algorithm with the objective function $f(x) = x^4 - 4x^2 - x$. Can you find choices for step-size and momentum that reliably perform well, i.e. converges towards the global minimum? If so, explain why you think these choices work well. If not, explain what obstacles you encounter and why they may be difficult to overcome.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

# objective function
def objective(x):
    return x**4 - 4*x**2 - x

# derivative of objective function
def derivative(x):
    return 4*x**3 - 8*x - 1

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size, momentum):
    # track all solutions
    solutions, scores = list(), list()
    # generate an initial point
    solution = bounds[:, 0] + np.random.rand(len(bounds)) * (bounds[:, 1] - bounds[
    # keep track of the change
    change = 0.0
    # run the gradient descent
    for i in range(n_iter):
        # calculate gradient
        gradient = derivative(solution)
        # calculate update
        new_change = step_size * gradient + momentum * change
        # take a step
        solution = solution - new_change
        # save the change
        change = new_change
        # evaluate candidate point
        solution_eval = objective(solution)
        # store solution
```

```
        solutions.append(solution)
        scores.append(solution_eval)
        # report progress
        print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
    return [solutions, scores]

# define range for input
bounds = np.asarray([[-2.0, 2.0]])
# define the total iterations
n_iter = 30

# Try different combinations of step size and momentum
step_size_values = [0.01, 0.1, 0.5]
momentum_values = [0.1, 0.3, 0.7, 0.9]

plt.figure(figsize=(15, 8))

for step_size in step_size_values:
    for momentum in momentum_values:
        # perform gradient descent
        solutions, scores = gradient_descent(objective, derivative, bounds, n_iter,
        # plot the solutions found
        label = f'Step Size={step_size}, Momentum={momentum}'
        plt.plot(solutions, scores, '.-', label=label)

# sample input range uniformly at 0.1 increments
inputs = np.arange(bounds[0, 0], bounds[0, 1] + 0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
plt.plot(inputs, results, label='Objective Function', linestyle='--', color='black'
# set plot labels and legend
plt.xlabel('Input')
plt.ylabel('Objective Function Value')
plt.legend()
# show the plot
plt.show()
```

```
>0 f([-1.26961691]) = -2.57978
>1 f([-1.28041805]) = -2.58960
>2 f([-1.2899633]) = -2.59714
>3 f([-1.29825466]) = -2.60281
>4 f([-1.30541765]) = -2.60703
>5 f([-1.31158408]) = -2.61016
>6 f([-1.3168772]) = -2.61245
>7 f([-1.32140936]) = -2.61414
>8 f([-1.32528161]) = -2.61536
>9 f([-1.3285839]) = -2.61625
>10 f([-1.33139563]) = -2.61690
>11 f([-1.33378642]) = -2.61737
>12 f([-1.33581691]) = -2.61770
>13 f([-1.33753968]) = -2.61794
>14 f([-1.33900012]) = -2.61812
>15 f([-1.3402373]) = -2.61824
>16 f([-1.3412847]) = -2.61833
>17 f([-1.34217098]) = -2.61840
>18 f([-1.34292058]) = -2.61844
>19 f([-1.34355435]) = -2.61847
>20 f([-1.34409002]) = -2.61850
>21 f([-1.34454265]) = -2.61851
>22 f([-1.34492503]) = -2.61853
>23 f([-1.345248]) = -2.61853
>24 f([-1.34552074]) = -2.61854
>25 f([-1.34575105]) = -2.61855
>26 f([-1.34594548]) = -2.61855
>27 f([-1.34610963]) = -2.61855
>28 f([-1.34624819]) = -2.61855
>29 f([-1.34636514]) = -2.61855
>0 f([-1.52805129]) = -2.35976
>1 f([-1.48604668]) = -2.47055
>2 f([-1.45106149]) = -2.53779
>3 f([-1.42443784]) = -2.57472
>4 f([-1.40479708]) = -2.59450
>5 f([-1.39039648]) = -2.60514
>6 f([-1.3797913]) = -2.61096
>7 f([-1.37191786]) = -2.61420
>8 f([-1.36602258]) = -2.61603
>9 f([-1.36157491]) = -2.61708
>10 f([-1.3581984]) = -2.61768
>11 f([-1.35562242]) = -2.61804
>12 f([-1.35364967]) = -2.61825
>13 f([-1.35213447]) = -2.61837
>14 f([-1.35096812]) = -2.61845
>15 f([-1.35006879]) = -2.61849
>16 f([-1.34937445]) = -2.61852
>17 f([-1.34883784]) = -2.61853
>18 f([-1.34842284]) = -2.61854
>19 f([-1.34810168]) = -2.61855
>20 f([-1.34785305]) = -2.61855
>21 f([-1.3476605]) = -2.61855
>22 f([-1.34751133]) = -2.61855
>23 f([-1.34739576]) = -2.61855
>24 f([-1.3473062]) = -2.61856
>25 f([-1.34723678]) = -2.61856
>26 f([-1.34718298]) = -2.61856
>27 f([-1.34714128]) = -2.61856
>28 f([-1.34710895]) = -2.61856
>29 f([-1.34708388]) = -2.61856
>0 f([-1.14452737]) = -2.37929
>1 f([-1.18244108]) = -2.45536
>2 f([-1.22744596]) = -2.52913
>3 f([-1.2731731]) = -2.58316
```

```
>4 f([-1.31448494]) = -2.61146
>5 f([-1.34771162]) = -2.61855
>6 f([-1.37087185]) = -2.61456
>7 f([-1.38370315]) = -2.60901
>8 f([-1.38740988]) = -2.60695
>9 f([-1.38417203]) = -2.60876
>10 f([-1.37656011]) = -2.61240
>11 f([-1.36701785]) = -2.61575
>12 f([-1.35751577]) = -2.61779
>13 f([-1.34939771]) = -2.61852
>14 f([-1.34338355]) = -2.61847
>15 f([-1.33966926]) = -2.61819
>16 f([-1.33806988]) = -2.61801
>17 f([-1.33816704]) = -2.61802
>18 f([-1.33943866]) = -2.61816
>19 f([-1.34136063]) = -2.61834
>20 f([-1.34347723]) = -2.61847
>21 f([-1.34544167]) = -2.61854
>22 f([-1.34703066]) = -2.61856
>23 f([-1.34813837]) = -2.61855
>24 f([-1.34875642]) = -2.61853
>25 f([-1.34894628]) = -2.61853
>26 f([-1.34881016]) = -2.61853
>27 f([-1.34846468]) = -2.61854
>28 f([-1.3480204]) = -2.61855
>29 f([-1.34756835]) = -2.61855
>0 f([1.79696743]) = -4.28630
>1 f([1.61284696]) = -5.25131
>2 f([1.41834794]) = -5.41821
>3 f([1.25263442]) = -5.06695
>4 f([1.135083]) = -4.62873
>5 f([1.07159512]) = -4.34623
>6 f([1.06096244]) = -4.29646
>7 f([1.0984995]) = -4.46918
>8 f([1.17714039]) = -4.79973
>9 f([1.28684379]) = -5.16849
>10 f([1.41328552]) = -5.41328
>11 f([1.53723142]) = -5.40540
>12 f([1.63645718]) = -5.17678
>13 f([1.69138017]) = -4.95046
>14 f([1.69257549]) = -4.94468
>15 f([1.64510091]) = -5.14616
>16 f([1.56589265]) = -5.36157
>17 f([1.47629264]) = -5.44409
>18 f([1.39505639]) = -5.39216
>19 f([1.33494691]) = -5.28746
>20 f([1.30248467]) = -5.21035
>21 f([1.29908258]) = -5.20150
>22 f([1.32225322]) = -5.25893
>23 f([1.36641641]) = -5.34875
>24 f([1.42342749]) = -5.42274
>25 f([1.48324879]) = -5.44324
>26 f([1.53522038]) = -5.40784
>27 f([1.5700779]) = -5.35372
>28 f([1.58223713]) = -5.32875
>29 f([1.57131581]) = -5.35132
>0 f([1.56897906]) = -5.35582
>1 f([1.41144761]) = -5.41139
>2 f([1.50010701]) = -5.43745
>3 f([1.45876961]) = -5.44238
>4 f([1.47994175]) = -5.44376
>5 f([1.46944867]) = -5.44408
>6 f([1.47477821]) = -5.44416
>7 f([1.47209394]) = -5.44418
```

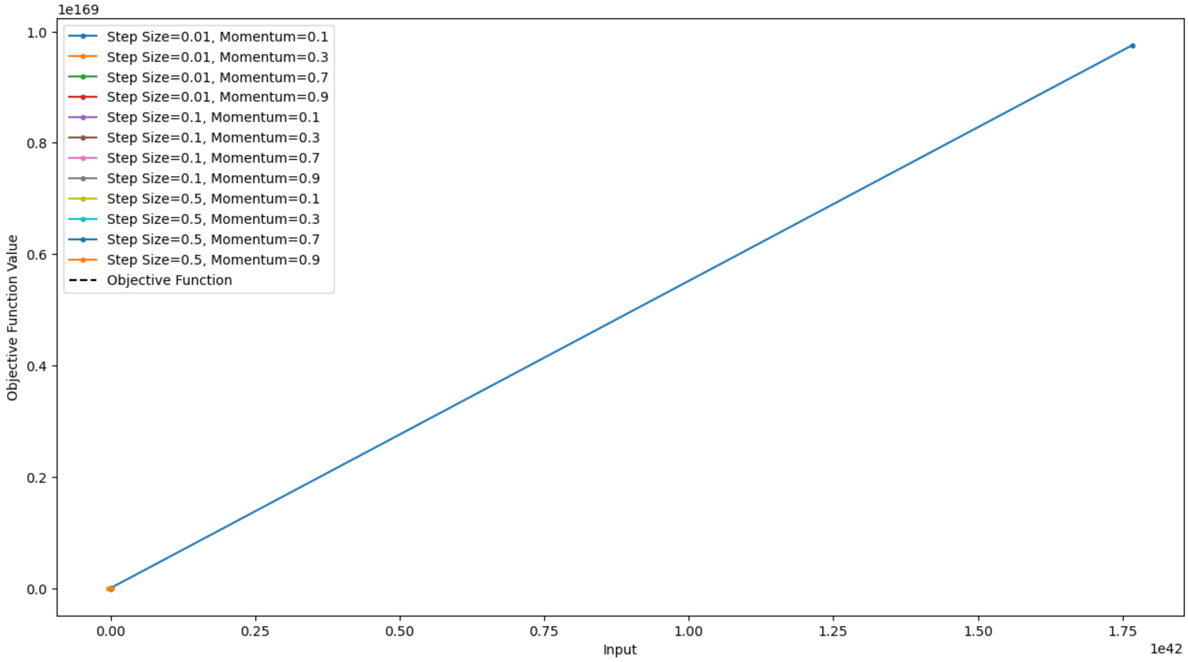
```
>8 f([1.47345398]) = -5.44419
>9 f([1.47276646]) = -5.44419
>10 f([1.47311451]) = -5.44419
>11 f([1.47293842]) = -5.44419
>12 f([1.47302754]) = -5.44419
>13 f([1.47298245]) = -5.44419
>14 f([1.47300527]) = -5.44419
>15 f([1.47299372]) = -5.44419
>16 f([1.47299957]) = -5.44419
>17 f([1.47299661]) = -5.44419
>18 f([1.4729981]) = -5.44419
>19 f([1.47299735]) = -5.44419
>20 f([1.47299773]) = -5.44419
>21 f([1.47299754]) = -5.44419
>22 f([1.47299763]) = -5.44419
>23 f([1.47299758]) = -5.44419
>24 f([1.47299761]) = -5.44419
>25 f([1.4729976]) = -5.44419
>26 f([1.4729976]) = -5.44419
>27 f([1.4729976]) = -5.44419
>28 f([1.4729976]) = -5.44419
>29 f([1.4729976]) = -5.44419
>0 f([-1.35817591]) = -2.61769
>1 f([-1.3563594]) = -2.61795
>2 f([-1.34277834]) = -2.61843
>3 f([-1.34448613]) = -2.61851
>4 f([-1.34844702]) = -2.61854
>5 f([-1.34763537]) = -2.61855
>6 f([-1.34651256]) = -2.61855
>7 f([-1.34684311]) = -2.61856
>8 f([-1.34715475]) = -2.61856
>9 f([-1.3470315]) = -2.61856
>10 f([-1.34694757]) = -2.61856
>11 f([-1.34699103]) = -2.61856
>12 f([-1.34701285]) = -2.61856
>13 f([-1.34699813]) = -2.61856
>14 f([-1.34699272]) = -2.61856
>15 f([-1.34699755]) = -2.61856
>16 f([-1.3469988]) = -2.61856
>17 f([-1.34699726]) = -2.61856
>18 f([-1.346997]) = -2.61856
>19 f([-1.34699749]) = -2.61856
>20 f([-1.34699752]) = -2.61856
>21 f([-1.34699738]) = -2.61856
>22 f([-1.34699738]) = -2.61856
>23 f([-1.34699742]) = -2.61856
>24 f([-1.34699742]) = -2.61856
>25 f([-1.3469974]) = -2.61856
>26 f([-1.34699741]) = -2.61856
>27 f([-1.34699741]) = -2.61856
>28 f([-1.34699741]) = -2.61856
>29 f([-1.34699741]) = -2.61856
>0 f([1.56729828]) = -5.35898
>1 f([1.59283682]) = -5.30433
>2 f([1.36849017]) = -5.35230
>3 f([1.38109526]) = -5.37252
>4 f([1.54106127]) = -5.40053
>5 f([1.52195852]) = -5.42188
>6 f([1.41599325]) = -5.41597
>7 f([1.43896468]) = -5.43398
>8 f([1.51439718]) = -5.42831
>9 f([1.48947099]) = -5.44172
>10 f([1.4418287]) = -5.43561
>11 f([1.46299228]) = -5.44330
```

```
>12 f([1.4956765]) = -5.43948
>13 f([1.47673649]) = -5.44407
>14 f([1.45671004]) = -5.44183
>15 f([1.47160167]) = -5.44417
>16 f([1.48454016]) = -5.44298
>17 f([1.47254207]) = -5.44419
>18 f([1.46496467]) = -5.44361
>19 f([1.47403537]) = -5.44418
>20 f([1.47851117]) = -5.44392
>21 f([1.47164579]) = -5.44418
>22 f([1.46927501]) = -5.44407
>23 f([1.4743053]) = -5.44418
>24 f([1.47546483]) = -5.44414
>25 f([1.47181568]) = -5.44418
>26 f([1.4713906]) = -5.44417
>27 f([1.47398698]) = -5.44418
>28 f([1.47401821]) = -5.44418
>29 f([1.47219739]) = -5.44419
>0 f([-1.2314529]) = -2.53475
>1 f([-1.6646007]) = -1.74112
>2 f([-1.44114045]) = -2.55295
>3 f([-1.09570494]) = -2.26521
>4 f([-1.03518904]) = -2.10291
>5 f([-1.26514576]) = -2.57533
>6 f([-1.57422964]) = -2.19710
>7 f([-1.45128714]) = -2.53743
>8 f([-1.17896827]) = -2.44889
>9 f([-1.12156549]) = -2.32774
>10 f([-1.30282438]) = -2.60558
>11 f([-1.52367659]) = -2.37290
>12 f([-1.4264437]) = -2.57235
>13 f([-1.2191113]) = -2.51693
>14 f([-1.18304812]) = -2.45648
>15 f([-1.33471074]) = -2.61753
>16 f([-1.48788604]) = -2.46640
>17 f([-1.3984969]) = -2.59955
>18 f([-1.24277569]) = -2.54974
>19 f([-1.2290646]) = -2.53142
>20 f([-1.35732641]) = -2.61782
>21 f([-1.45836319]) = -2.52555
>22 f([-1.37531458]) = -2.61291
>23 f([-1.26026488]) = -2.57021
>24 f([-1.26427691]) = -2.57444
>25 f([-1.37098315]) = -2.61452
>26 f([-1.43304817]) = -2.56408
>27 f([-1.35816662]) = -2.61769
>28 f([-1.27518786]) = -2.58501
>29 f([-1.291222]) = -2.59806
>0 f([-0.82514149]) = -1.43472
>1 f([-2.55729411]) = 19.16665
>2 f([20.98845868]) = 192270.76555
>3 f([-18383.68157399]) = 114216790061015984.00000
>4 f([1.24258887e+13]) = 23840196837028178466698665212017276287171706501464064.000
00
>5 f([-3.83718178e+39]) = 21679511553836297908596895911200002883580032129468217714
5156493359036470748656332998415683643217082911604418256207224508218957932794615089
272084711578772963328.00000
>6 f([1.12997053e+119]) = inf
>7 f([-inf]) = nan
>8 f([nan]) = nan
>9 f([nan]) = nan
>10 f([nan]) = nan
>11 f([nan]) = nan
>12 f([nan]) = nan
```

```
>13 f([nan]) = nan
>14 f([nan]) = nan
>15 f([nan]) = nan
>16 f([nan]) = nan
>17 f([nan]) = nan
>18 f([nan]) = nan
>19 f([nan]) = nan
>20 f([nan]) = nan
>21 f([nan]) = nan
>22 f([nan]) = nan
>23 f([nan]) = nan
>24 f([nan]) = nan
>25 f([nan]) = nan
>26 f([nan]) = nan
>27 f([nan]) = nan
>28 f([nan]) = nan
>29 f([nan]) = nan
>0 f([-1.60517262]) = -2.06238
>1 f([-0.26374209]) = -0.00966
>2 f([-0.37958955]) = -0.17600
>3 f([-1.32331324]) = -2.61476
>4 f([-1.76502237]) = -0.99107
>5 f([2.53953775]) = 13.25631
>6 f([-18.26718084]) = 110032.45428
>7 f([12094.06943178]) = 21393892948245596.00000
>8 f([-3.53791472e+12]) = 156671298105457306410074601896920853143141297946624.0000
0
>9 f([8.85670291e+37]) = 6153019735446853667091640880149256410779279524363966454647386386418970020829665013280678339166194545353912159766877764781527084070601969530
0839095664640.00000
>10 f([-1.38946057e+114]) = inf
>11 f([inf]) = nan
>12 f([nan]) = nan
>13 f([nan]) = nan
>14 f([nan]) = nan
>15 f([nan]) = nan
>16 f([nan]) = nan
>17 f([nan]) = nan
>18 f([nan]) = nan
>19 f([nan]) = nan
>20 f([nan]) = nan
>21 f([nan]) = nan
>22 f([nan]) = nan
>23 f([nan]) = nan
>24 f([nan]) = nan
>25 f([nan]) = nan
>26 f([nan]) = nan
>27 f([nan]) = nan
>28 f([nan]) = nan
>29 f([nan]) = nan
>0 f([2.75043646]) = 24.21768
>1 f([-26.32548803]) = 477546.61069
>2 f([36337.29484907]) = 1743453505859556096.00000
>3 f([-9.59594551e+13]) = 8479126086404141560236365745479073105290421163414205235
2.00000
>4 f([1.76723098e+42]) = 975378689599623778448671947805686841099746111134960716591
89520168302525810563555371208227438249547880930152859415367543520255138962023876091690104068094886405220094443 52.00000
>5 f([-1.1038497e+127]) = inf
>6 f([inf]) = nan
>7 f([nan]) = nan
>8 f([nan]) = nan
>9 f([nan]) = nan
>10 f([nan]) = nan
```

```
>11 f([nan]) = nan
>12 f([nan]) = nan
>13 f([nan]) = nan
>14 f([nan]) = nan
>15 f([nan]) = nan
>16 f([nan]) = nan
>17 f([nan]) = nan
>18 f([nan]) = nan
>19 f([nan]) = nan
>20 f([nan]) = nan
>21 f([nan]) = nan
>22 f([nan]) = nan
>23 f([nan]) = nan
>24 f([nan]) = nan
>25 f([nan]) = nan
>26 f([nan]) = nan
>27 f([nan]) = nan
>28 f([nan]) = nan
>29 f([nan]) = nan
>0 f([-2.2143156]) = 6.64285
>1 f([9.7399711]) = 8610.54659
>2 f([-1788.04568317]) = 10221482717250.99805
>3 f([1.14331375e+10]) = 170868380943844796712405772758546508677 12.00000
>4 f([-2.98900247e+30]) = 798187818600198917600635006153108578036557614263120757 65
673555345022403858356775036223457007001890166110580510182044860416.00000
>5 f([5.34083077e+91]) = inf
>6 f([-3.0468877e+275]) = nan
>7 f([inf]) = nan
>8 f([nan]) = nan
>9 f([nan]) = nan
>10 f([nan]) = nan
>11 f([nan]) = nan
>12 f([nan]) = nan
>13 f([nan]) = nan
>14 f([nan]) = nan
>15 f([nan]) = nan
>16 f([nan]) = nan
>17 f([nan]) = nan
>18 f([nan]) = nan
>19 f([nan]) = nan
>20 f([nan]) = nan
>21 f([nan]) = nan
>22 f([nan]) = nan
>23 f([nan]) = nan
>24 f([nan]) = nan
>25 f([nan]) = nan
>26 f([nan]) = nan
>27 f([nan]) = nan
>28 f([nan]) = nan
>29 f([nan]) = nan
```

```
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\2263159172.py:6: RuntimeWarnin
g: overflow encountered in power
  return x**4 - 4*x**2 - x
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\2263159172.py:10: RuntimeWarnin
g: overflow encountered in power
  return 4*x**3 - 8*x - 1
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\2263159172.py:6: RuntimeWarnin
g: invalid value encountered in subtract
  return x**4 - 4*x**2 - x
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\2263159172.py:10: RuntimeWarnin
g: invalid value encountered in subtract
  return 4*x**3 - 8*x - 1
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\2263159172.py:6: RuntimeWarnin
g: overflow encountered in square
  return x**4 - 4*x**2 - x
```



Gradient descent for functions of two variables

Task 4 - Define delta_w in the code below for gradient descent in the two-variable case.

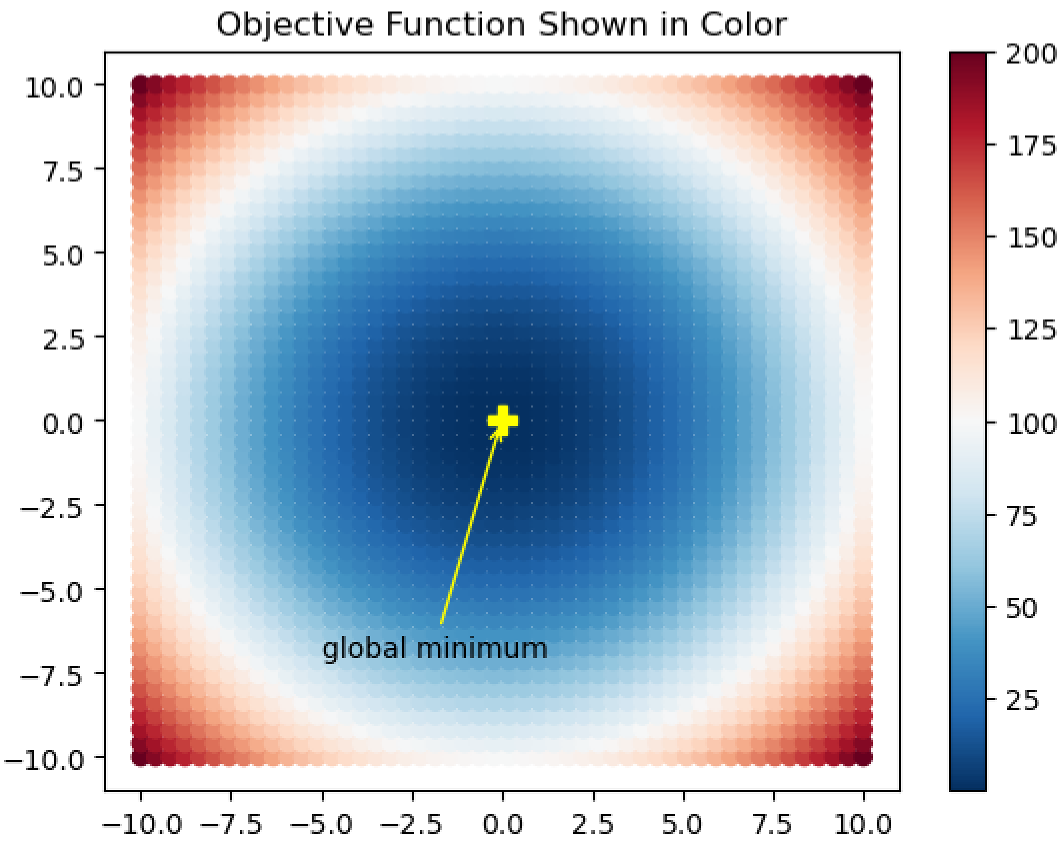
In [10]:
```
import numpy as np

def gradient_descent(max_iterations, threshold, w_init,
                     obj_func, grad_func, extra_param=[],
                     learning_rate=0.05, momentum=0.8):

    w = w_init
    w_history = w
    f_history = obj_func(w, extra_param)
    delta_w = np.zeros_like(w)  # Initialize delta_w
    i = 0
    diff = 1.0e10

    while i < max_iterations and diff > threshold:
        # Calculate gradient
        gradient = grad_func(w, extra_param)

        # Update delta_w using momentum
        delta_w = -learning_rate * gradient + momentum * delta_w

        w = w + delta_w
```

```
        # store the history of w and f
        w_history = np.vstack((w_history, w))
        f_history = np.vstack((f_history, obj_func(w, extra_param)))

        # update iteration number and diff between successive values
        # of the objective function
        i += 1
        diff = np.absolute(f_history[-1] - f_history[-2])

    return w_history, f_history
```

Objective function $f(x,y) = x^2 + y^2$

In [11]:

```
def visualize_fw():
    xcoord = np.linspace(-10.0,10.0,50)
    ycoord = np.linspace(-10.0,10.0,50)
    w1,w2 = np.meshgrid(xcoord,ycoord)
    pts = np.vstack((w1.flatten(),w2.flatten()))

    # All 2D points on the grid
    pts = pts.transpose()

    # Function value at each point
    f_vals = np.sum(pts*pts,axis=1)
    function_plot(pts,f_vals)
    plt.title('Objective Function Shown in Color')
    plt.show()
    return pts,f_vals

# Helper function to annotate a single point
def annotate_pt(text,xy,xytext,color):
    plt.plot(xy[0],xy[1],marker='P',markersize=10,c=color)
    plt.annotate(text,xy=xy,xytext=xytext,
                 # color=color,
                 arrowprops=dict(arrowstyle="->",
                 color = color,
                 connectionstyle='arc3'))

# Plot the function
# Pts are 2D points and f_val is the corresponding function value
def function_plot(pts,f_val):
    f_plot = plt.scatter(pts[:,0],pts[:,1],
                         c=f_val,vmin=min(f_val),vmax=max(f_val),
                         cmap='RdBu_r')
    plt.colorbar(f_plot)
    # Show the optimal point
    annotate_pt('global minimum',(0,0),(-5,-7),'yellow')

pts,f_vals = visualize_fw()
```



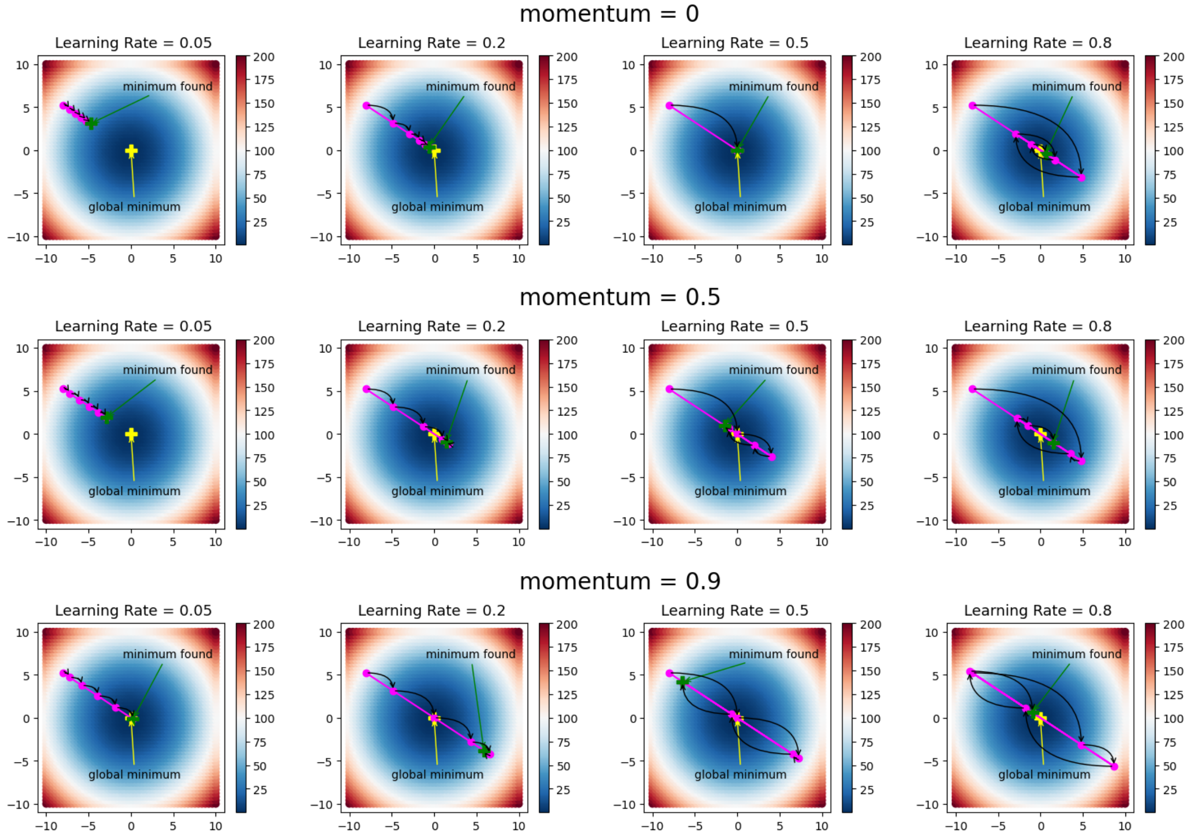
In [12]:

```
# Objective function
def f(w,extra=[]):
    return np.sum(w*w)

# Function to compute the gradient
def grad(w,extra=[]):
    return 2*w

# Function to plot the objective function
# and learning history annotated by arrows
# to show how learning proceeded
def visualize_learning(w_history):

    # Make the function plot
    function_plot(pts,f_vals)

    # Plot the history
    plt.plot(w_history[:,0],w_history[:,1],marker='o',c='magenta')

    # Annotate the point found at last iteration
    annotate_pt('minimum found',
                (w_history[-1,0],w_history[-1,1]),
                (-1,7),'green')
    iter = w_history.shape[0]
    for w,i in zip(w_history,range(iter-1)):
        # Annotate with arrows to show history
        plt.annotate("",
                     xy=w, xycoords='data',
                     xytext=w_history[i+1,:], textcoords='data',
                     arrowprops=dict(arrowstyle='<-',
                                     connectionstyle='angle3'))

def solve_fw():
    # Setting up
    rand = np.random.RandomState(19)
    w_init = rand.uniform(-10,10,2)
```

```
    fig, ax = plt.subplots(nrows=4, ncols=4, figsize=(18, 12))
    learning_rates = [0.05,0.2,0.5,0.8]
    momentum = [0,0.5,0.9]
    ind = 1

    # Iteration through all possible parameter combinations
    for alpha in momentum:
        for eta,col in zip(learning_rates,[0,1,2,3]):
            plt.subplot(3,4,ind)
            w_history,f_history = gradient_descent(5,-1,w_init, f,grad,[],eta,alpha

            visualize_learning(w_history)
            ind = ind+1
            plt.text(-9, 12,'Learning Rate = '+str(eta),fontsize=13)
            if col==1:
                plt.text(10,15,'momentum = ' + str(alpha),fontsize=20)

    fig.subplots_adjust(hspace=0.5, wspace=.3)
    plt.show()
```

In [13]:
```
solve_fw()
```

```
C:\Users\Akshay\AppData\Local\Temp\ipykernel_62112\1505662540.py:45: MatplotlibDep
recationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will
be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(3,4,ind)
```



Task 5 (OPTIONAL) - Apply this to other objective functions of your choosing, for example use some $f(x, y)$ which has both local and global minima, and see if you can find choices for step-size and momentum which achieve reliabl