Optimisation As A Programming Tool

Luís Pedro Coelho

Programming for Scientists

February 24, 2009

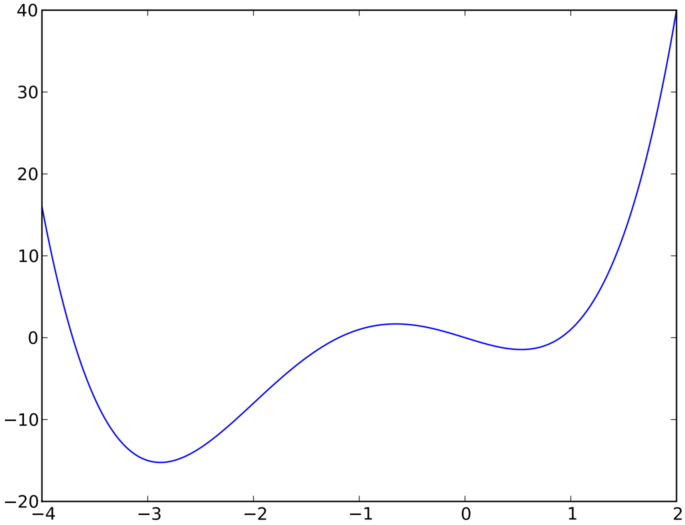
Optimisation

$$\min_x f(x)$$
$$s.t.\ x \in S$$

Simple Example

$$f(x) = x^4 + 4x^3 - 4x$$

Let's look to minimise $f(x)$.


40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2

Taxonomy of Minimisation Problems

$$\min_x f(x)$$
$$s.t.\ x \in S$$

- What is the form of f?
- What is the form of S?

Linear Least-Squares

$$\min_{\beta}(y - X\beta)^2;$$

or find an approximation to

$$y = X\beta,$$

where $y \in R^n \beta \in R^m, X \in R^{n \times m}$, where y and X are given.
Linear least-squares has a closed-form fast solution.

Two Solutions

Pseudo-Inverse

$$\hat{\beta} = (XX^T)^{-1}X^T y$$

This is, generally, not a good idea.

Two Solutions

Pseudo-Inverse

$$\hat{\beta} = (XX^T)^{-1}X^Ty$$

This is, generally, not a good idea.

Use An Equation Solver

```
X = np.array(...)
y = np.array(...)
beta = scipy.linalg.leastsq(X,y)
```

Polynomial Fit

Let's say you have one input variable x and an output y and you want to fit a 3rd-degree polynomial.
Is this a linear regression?

Polynomial Fit

Let's say you have one input variable x and an output y and you want to fit a 3rd-degree polynomial.
Is this a linear regression? Yes!

$$y = \beta_3 x^3 + \beta_2 x^2 + \beta_1 x + \beta_0,$$

can be written as

$$y = [x^3 x^2 x^1 1][\beta_3 \beta_2 \beta_1 \beta_0]^T.$$

Linear Programming

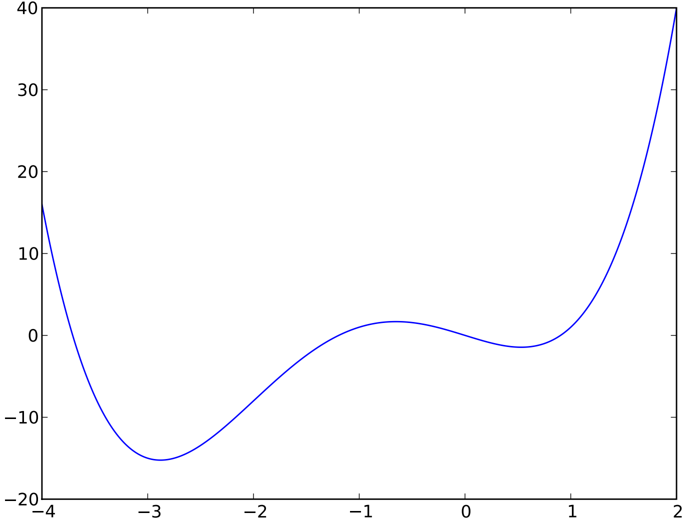
$$\min c \cdot x$$
$$s.t. \;\; Ax \leq b$$

Example

You can manufacture 3 types of widgets:

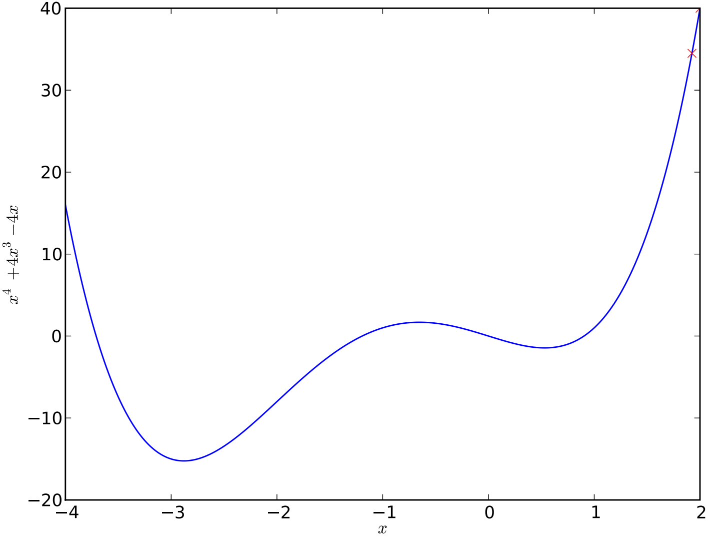
	R_1	R_2	R_3	R_4	P
W_1	10	20	9	0.1	20
W_2	5	1	10	1	3
W_3	100	120	9	1.8	90

Gradient Descent

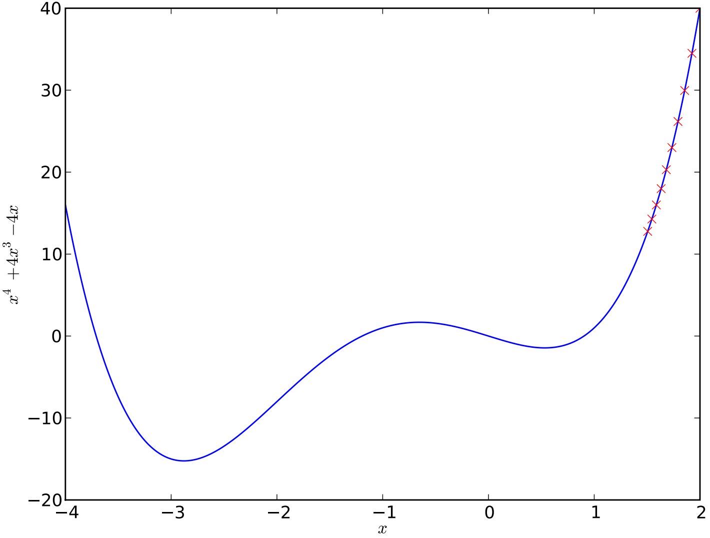


Gradient Descent

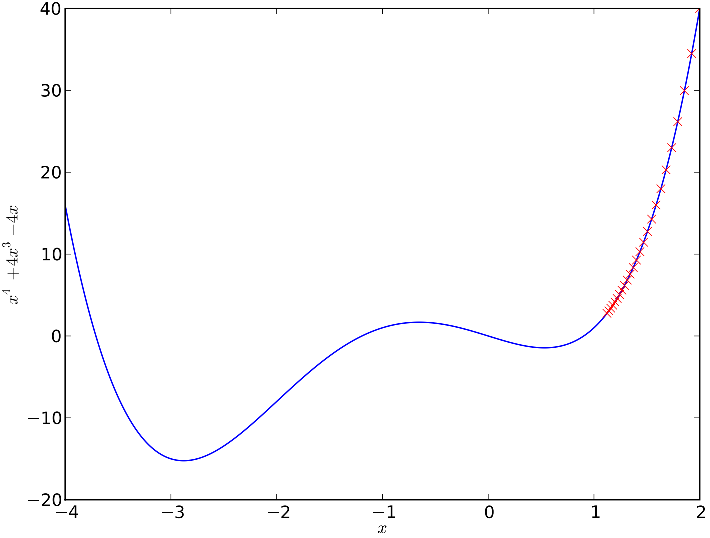
1. Start with some prediction x_0
2. $i \leftarrow 0$
3. While *not bored*
    1. $x_{i+1} \leftarrow x_i - \varepsilon f'(x_i)$
    2. $i \leftarrow i + 1$



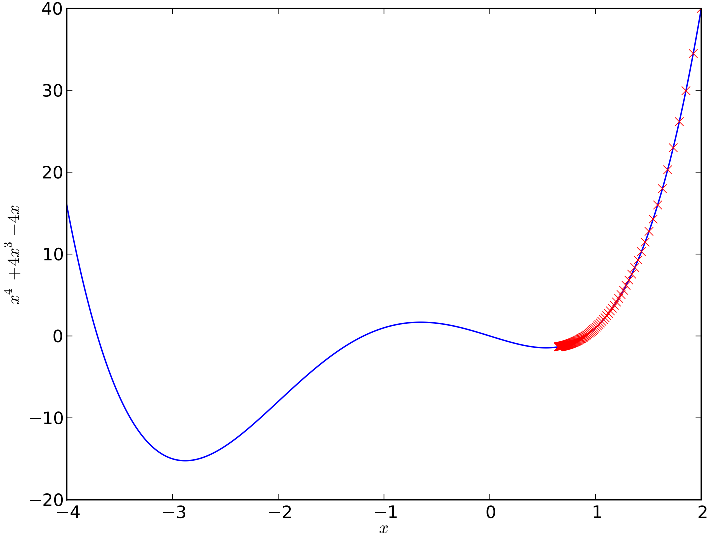
40
30
20
10
0
−10
−20
x^4+4x^3-4x
−4
−3
−2
−1
0
1
2
x



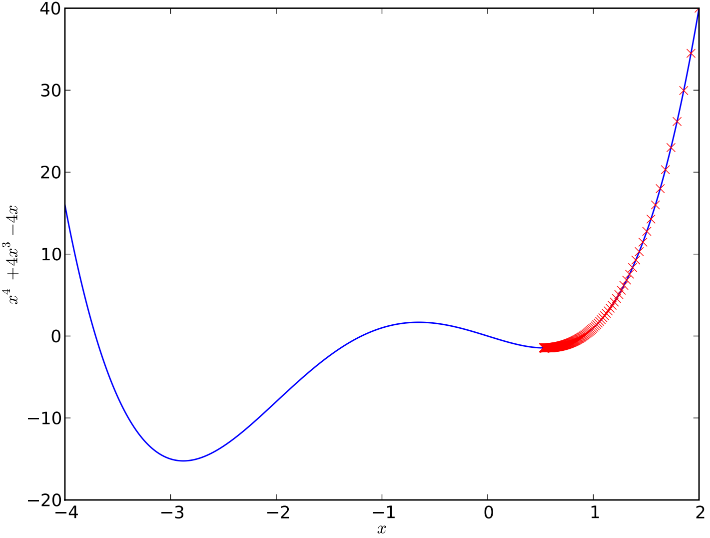
40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

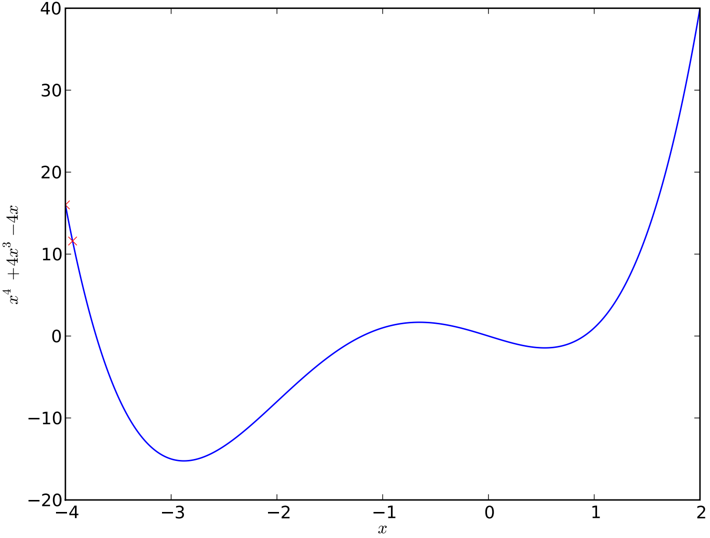


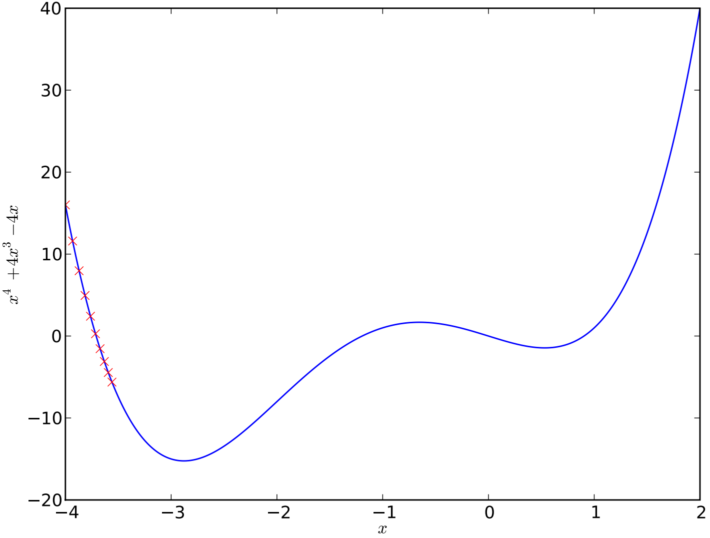
40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$



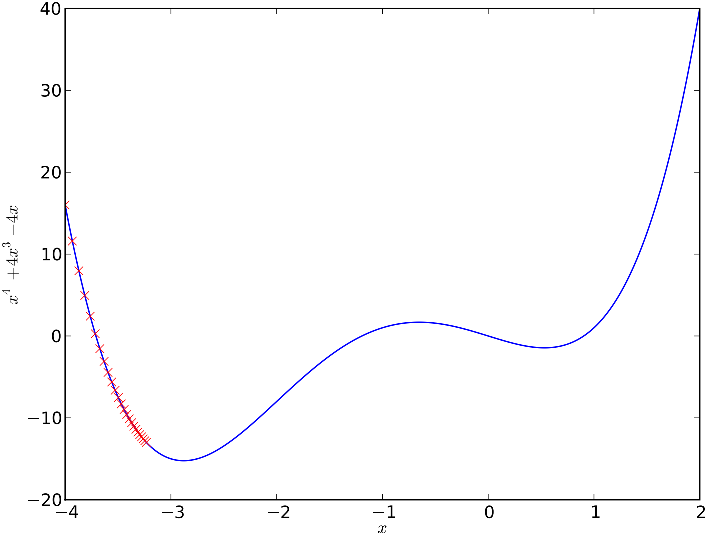
40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

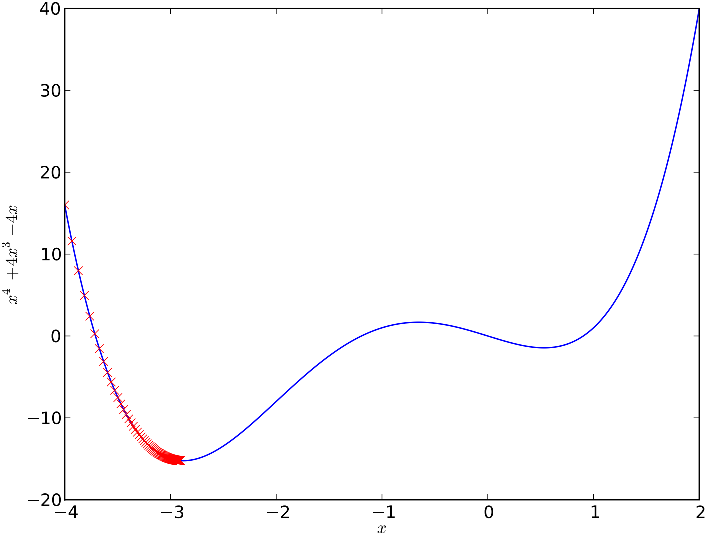

40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$


40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

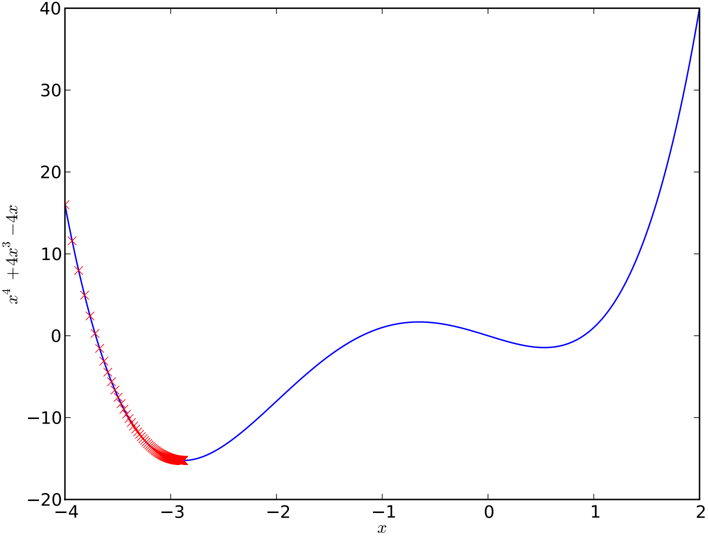


40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$


40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

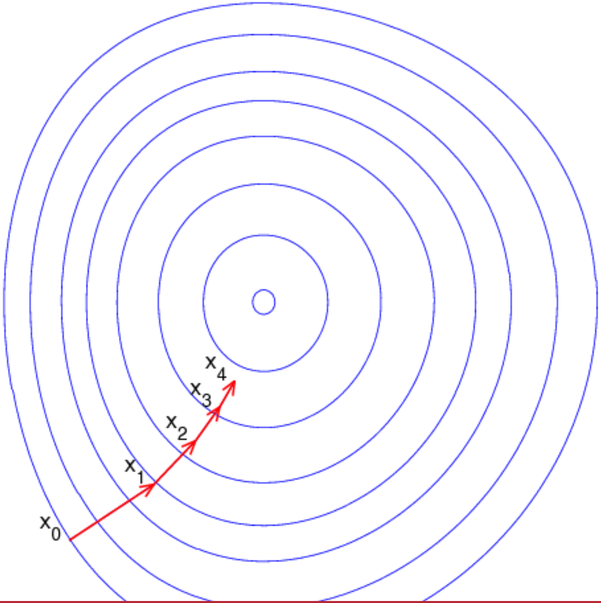


40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
x^4+4x^3-4x



40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

Multi-dimension



Newton's Method

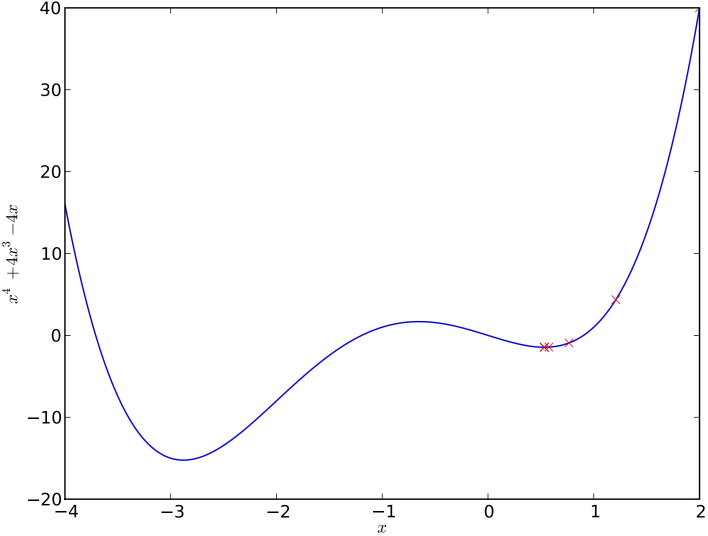
Newton-Raphson solves a similar problem:

$$f(x) = 0$$

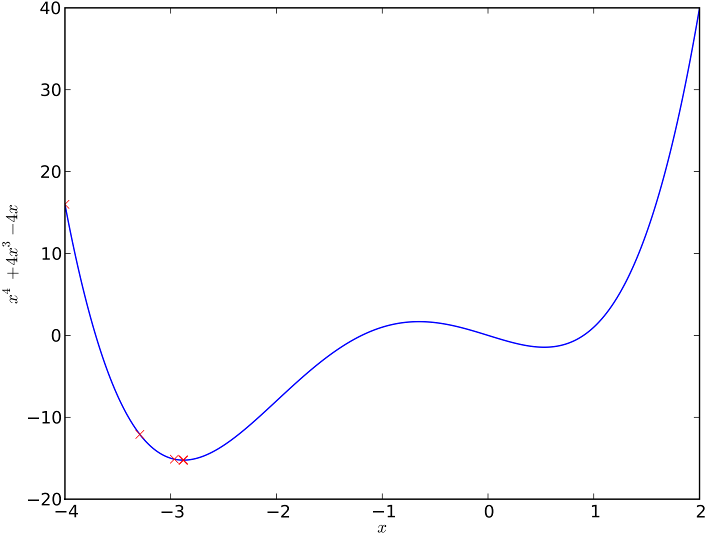
by iterating

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

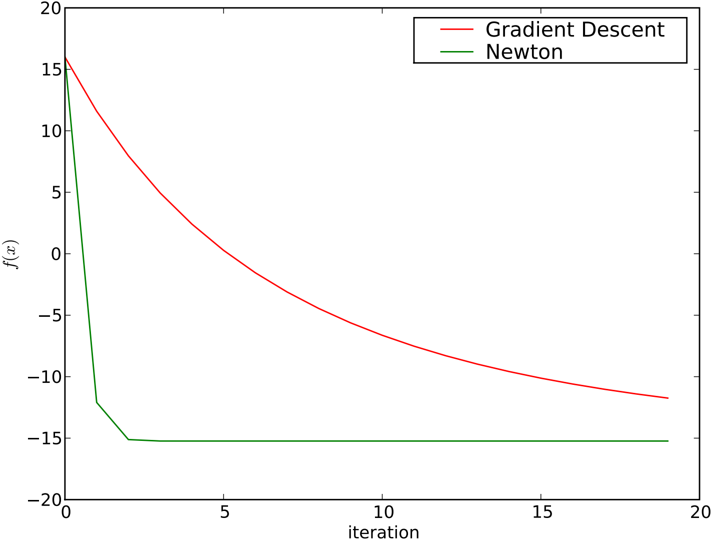
Newton's Method



Newton's Method



Newton's Method vs. Gradient Descent



Derivatives?

In gradient descent, we need derivatives. What if the function is a complex function?

Derivatives?

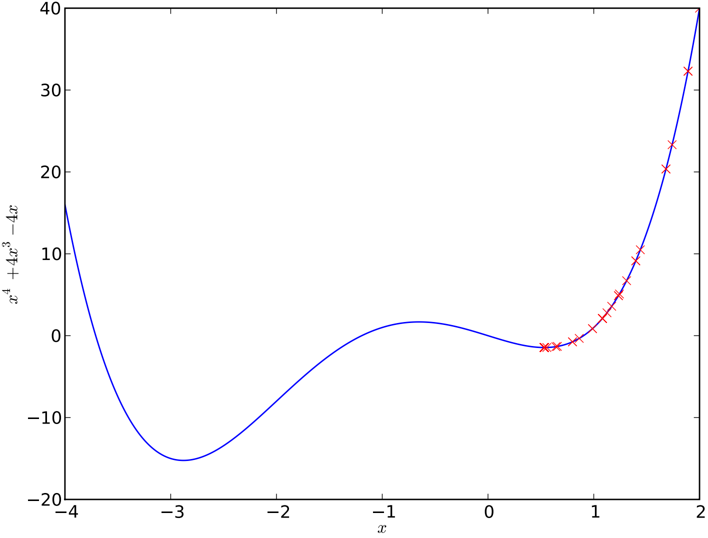
In gradient descent, we need derivatives. What if the function is a complex function?

$$\frac{\partial f}{\partial x_i}(\vec{x}) = \frac{f(x_0, \cdots, x_i + h, \cdots, x_n) - f(x_0, \cdots, x_i, \cdots, x_n)}{h}.$$

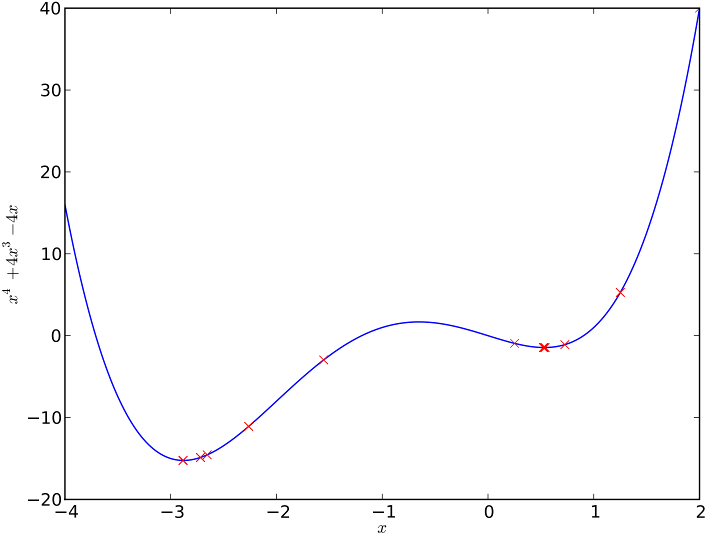
This works for very complex functions.

Random Greedy Hill Descent

1. $x_0 \leftarrow ...$
2. For $i \in \{1, \cdots, N\}$
    1. $C \leftarrow x_i + \mathcal{N}(0, \sigma)$
    2. If $f(C) < f(x_i)$, then $x_{i+1} \leftarrow C$
    3. Else $x_{i+1} \leftarrow x_i$



40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$



40
30
20
10
0
−10
−20
−4
−3
−2
−1
0
1
2
x
$x^4 + 4x^3 - 4x$

In Practice

Always Use Pre-Written Functions!

- They are (mostly) gradient descent or Newton-Raphson.
- However, they are much better than anything you could write (unless you spend a couple of years working on it).
- They can fall into local minima.

```python
import scipy.optimize
def f(x):
    return x**4+4*x**3-4*x

print scipy.optimize.fmin(f,2.)
print scipy.optimize.fmin_cg(f,2.)
```

prints out

```
Optimization terminated successfully.
         Current function value: -1.445622
         Iterations: 17
         Function evaluations: 34
array([ 0.53212891])

Optimization terminated successfully.
         Current function value: -15.234422
         Iterations: 4
         Function evaluations: 30
         Gradient evaluations: 10
array([-2.87938508])
```

OpenOpt

```
import scikits.openopt
def f(x):
    return x**4+4*x**3-4*x

P = scikits.openopt.NLP(f,2)
result = P.solve('ralg')
print 'result:', result.xf
```

```
solver: ralg   problem: unnamed   goal: minimum
 iter    objFunVal
   ...
istop:  4 (|| F[k] - F[k-1] || < ftol)
Solver:   Time Elapsed = 0.03   CPU Time Elapsed = 0.0
objFunValue: -15.234422

result: [-2.87941895]
```