» More On Approximate Derivatives

* Recall our general iterative minimisation algorithm:

```
x=x0
for k in range(num_iters):
    step = calcStep(fn,x)
    x = x - step
```

e.g. with $step = \alpha[\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x)]$

* In SGD:
  * We use approximate derivatives $DJ_{x_1}(\theta)$ etc instead of exact derivatives $\frac{\partial f}{\partial x_1}(x)$ etc
  * Need cost function of form

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} loss(\theta, x^{(i)}, y^{(i)})$$

and use approx of form

$$DJ_{\theta_1}(\theta) = \frac{1}{|N|}\sum_{i \in N} \frac{\partial loss}{\partial \theta_1}(\theta, x^{(i)}, y^{(i)}$$

with $N \subseteq \{1, 2, \ldots, m\}$

» Finite Difference Approximation to Derivative

* Recall

$$f(x) \approx f(x') + \frac{\partial f}{\partial x_1}(x')(x_1 - x_1') + \frac{\partial f}{\partial x_2}(x')(x_2 - x_2') + \cdots + \frac{\partial f}{\partial x_n}(x')(x_n - x_n')$$

* Select $x = x'$ and then add a small amount δ to element 1 of x i.e.

$$x = [x_1' + \delta, x_n', \ldots, x_n']$$

Then

$$f(x) \approx f(x') + \frac{\partial f}{\partial x_1}(x')\delta$$

i.e.

$$\frac{\partial f}{\partial x_1}(x') \approx \frac{f(x) - f(x')}{\delta} = \frac{f(x' + \delta) - f(x')}{\delta}$$

$\rightarrow$ *finite difference approximation to derivative*

* Perturbation δ needs to be small e.g. 0.01 or less.

» Finite Difference Approximation to Derivative

Example:

* $f(x) = x^2$, $\frac{df}{dx}(x) = 2x$
* At point $x' = 1$ then $\frac{df}{dx}(1) = 2.0$
* Finite difference:

$$\frac{f(x' + \delta) - f(x')}{\delta} = \frac{f(1 + \delta) - f(1)}{\delta} = \frac{(1 + \delta)^2 - 1}{\delta}$$

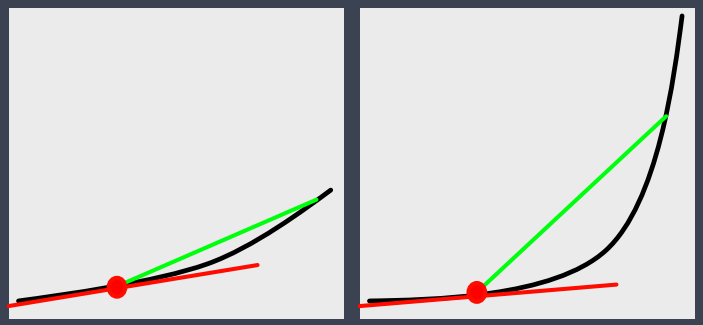
For $\delta = 0.1 \rightarrow$ 2.1
For $\delta = 0.01 \rightarrow$ 2.01
For $\delta = 0.001 \rightarrow$ 2.0010

» Finite Difference Approximation to Derivative

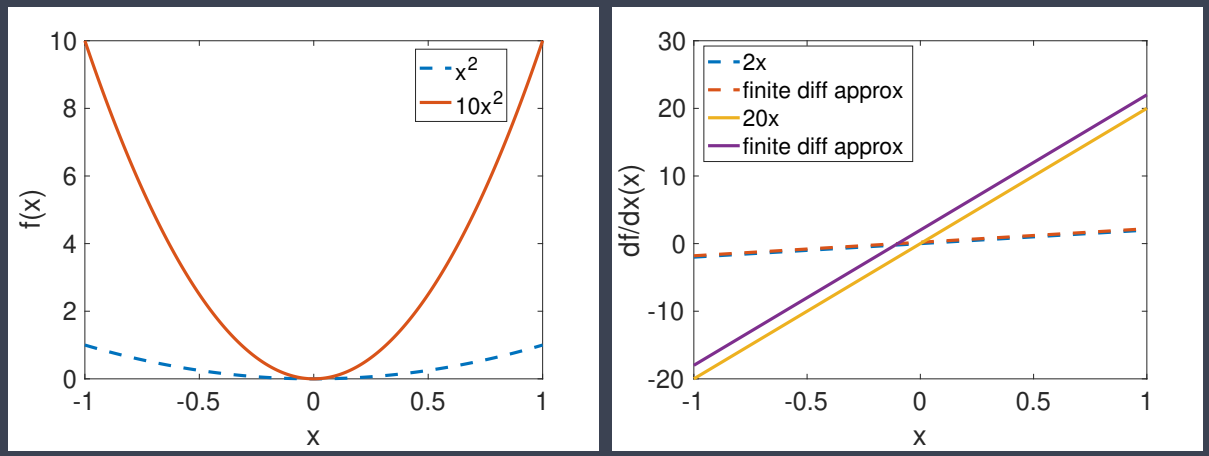
- We can use finite difference approx in any of our gradient descent algorithms and in SGD
- An example of a *gradient-free* optimisation method
- To calc approx derivative requires two evaluations of function $f(x)$ for every element of x, so $2n$ function evaluations at each step $\rightarrow$ may be more expensive then exact derivative calc, but *handy when exact derivatives are not available or not easy to calculate*
- Aside from computation cost, main issue is accuracy of the approximation $\rightarrow$ depends on δ, how to choose that?

» Finite Difference Approximation to Derivative

* Main issue is accuracy of the approximation $\rightarrow$ depends on δ, how to choose that?
* As the curvature of $f(x)$ increases, the error in the finite difference approximation increases.



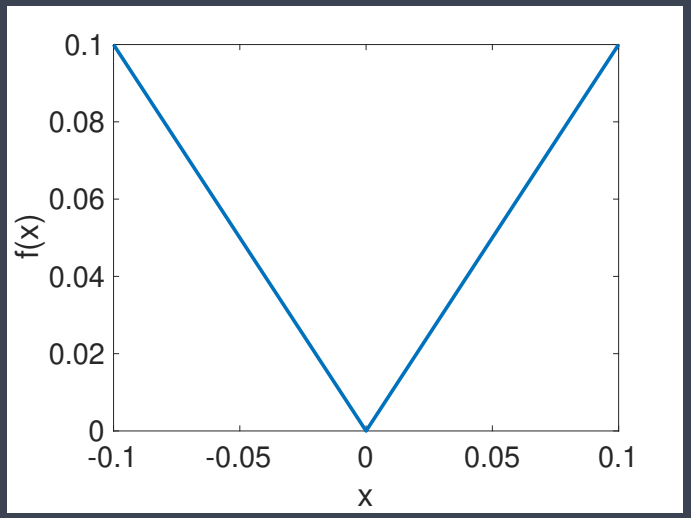
* Example: $f(x) = x^2$, $\frac{df}{dx}(x) = 2x$ and $f(x) = 10x^2$, $\frac{df}{dx}(x) = 20x$. $\delta = 0.2$

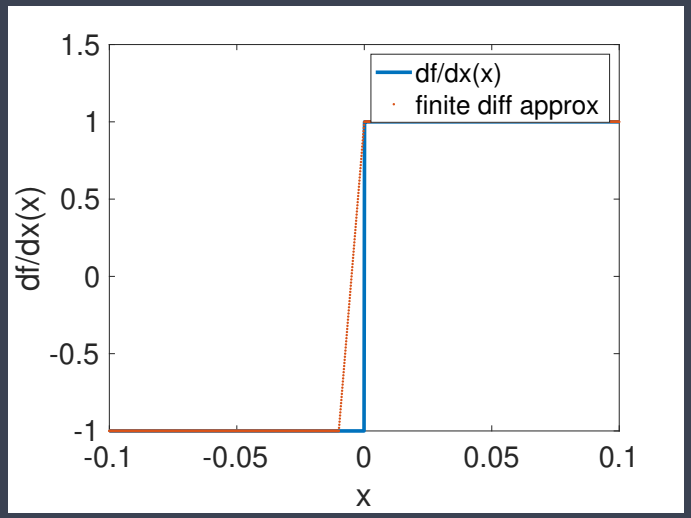


* Need to adjust δ to be small in regions where curvature is large, otherwise will finite diff approx to derivative might have large errors.

» Finite Difference Approximation to Derivative

* Example: $f(x) = |x|$, $\frac{df}{dx}(x) = sign(x)$. $\delta = 0.01$

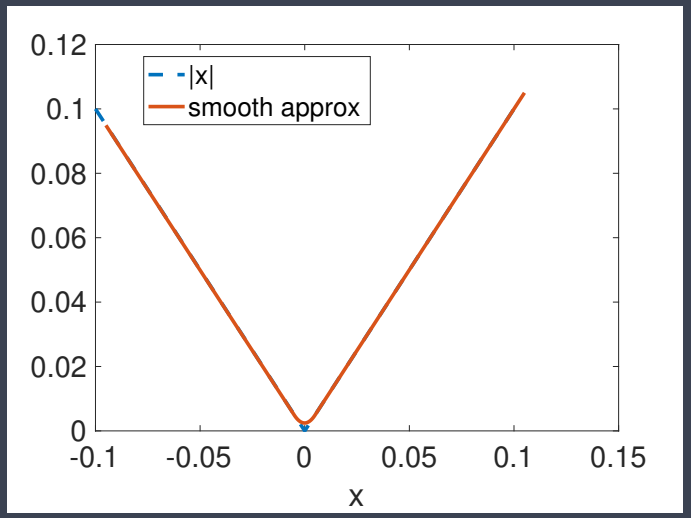




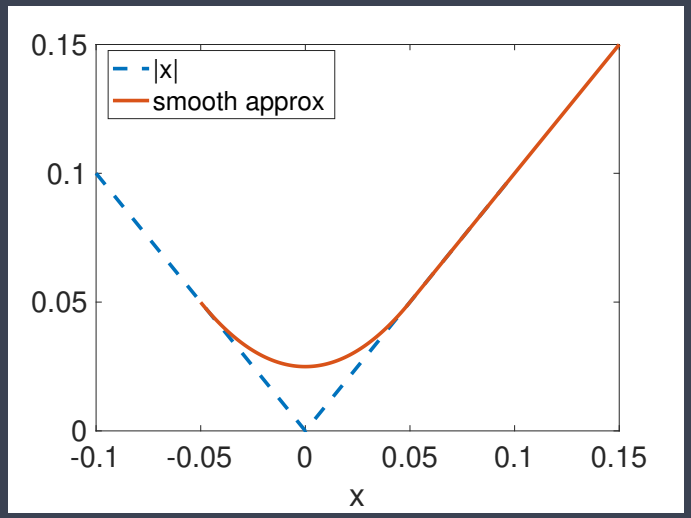
* Curvature is effectively infinite at $x = 0 \rightarrow$ need to make δ v small (close to zero).
* *Another way to think about it: finite different approx is the exact derivative of a different function $\hat{f}(x)$ ...*

» Finite Difference Approximation to Derivative

* Another way to think about it: finite different approx is the exact derivative of a different function $\hat{f}(x)$ ...



$\delta = 0.01$



$\delta = 0.1$

* Exact derivative of $\hat{f}(x)$ = finite difference approx derivative of $|x|$
* See that $\hat{f}(x)$ rounds off the kink in $|x|$, the amount of rounding depending on $\delta \rightarrow$ as δ is made smaller $\hat{f}(x)$ more closely approximates $|x|$
* *So when using finite difference approx in optimisation, we'll find the min of $\hat{f}(x)$ rather than $f(x)$. So long as $\hat{f}(x)$ is close to $f(x)$ that's fine.*

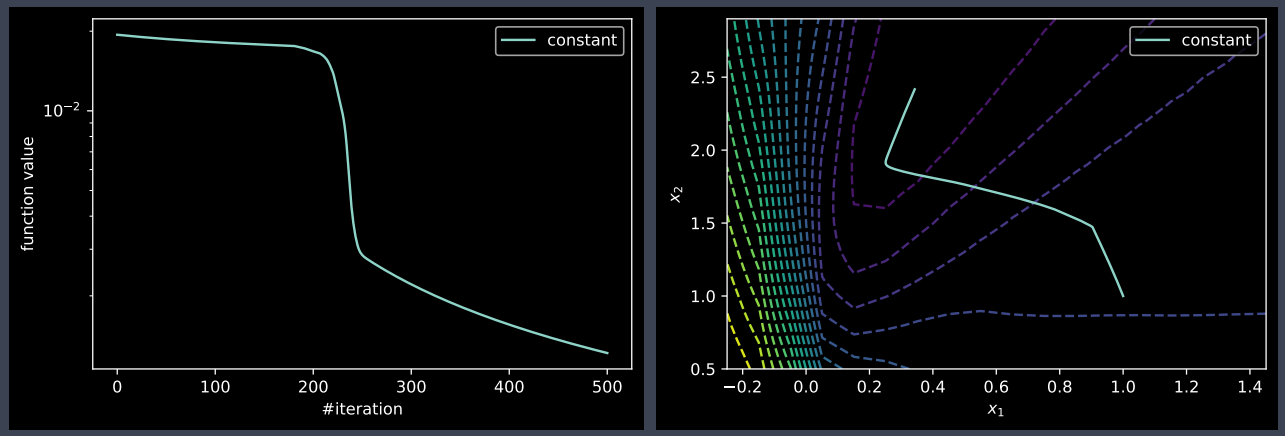
» Finite Difference Approximation to Derivative

* Matlab code for previouus example:

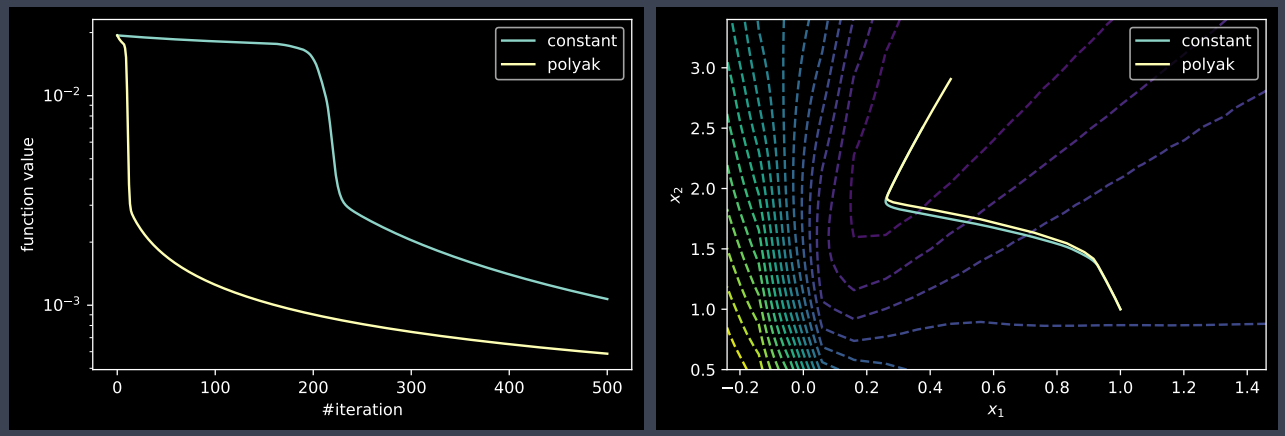
```
dd=0.0001;
x=[-0.1:dd:0.1];
d=0.1;
df=(abs(x+d)-abs(x))/d;
plot(x,sign(x),x,df,'.','LineWidth',3)

% function corresponding to finite diff approx
ff=abs(x(1))+dd*cumsum(df)-d/2;
plot(x,abs(x),'--',x+d/2,ff,'LineWidth',3)
```

» Example

* Toy neural net, starting point $x = [1, 1]$, constant $\alpha_0 = 0.75$
* Gradient descent:



exact derivatives

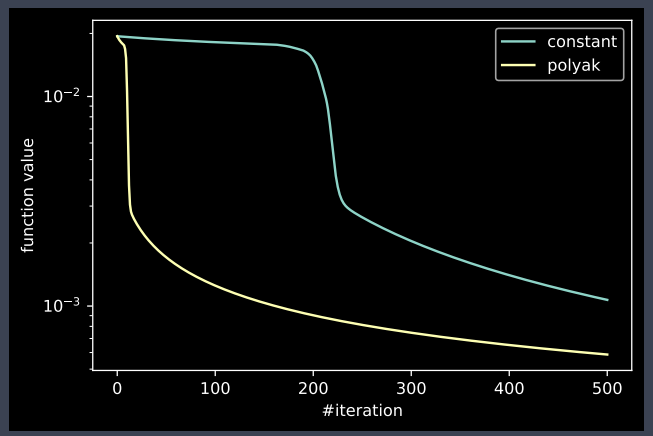


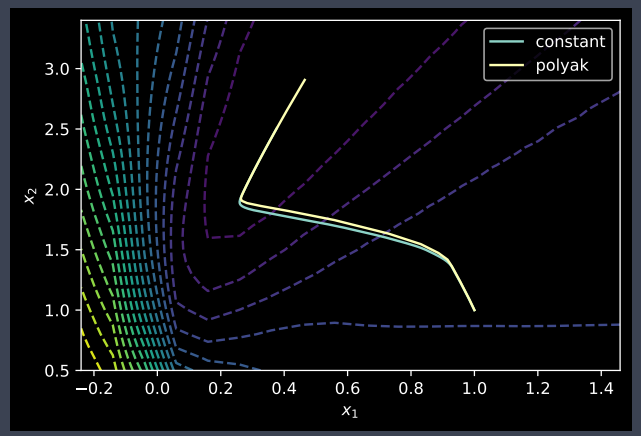
finite difference approx $\delta = 0.1$

* Performance is pretty similar!

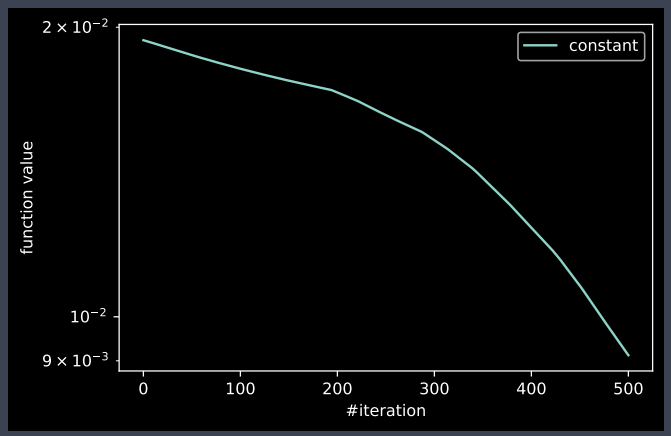
» Example

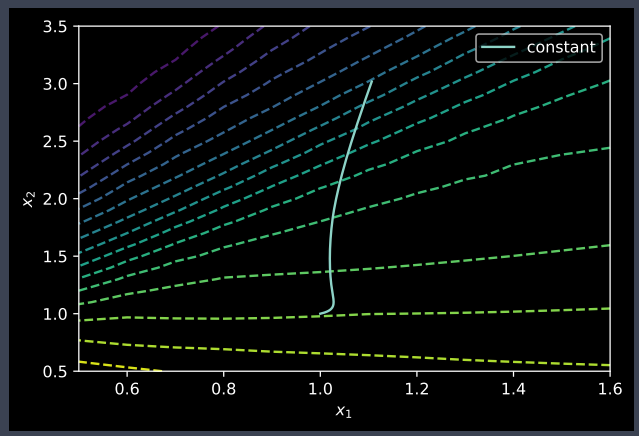
* Toy neural net, starting point $x = [1, 1]$, constant $\alpha_0 = 0.75$
* Gradient descent:





finite difference approx $\delta = 0.1$





finite difference approx $\delta = 1$

* $\delta = 1$ is too big and leads to a poorer solution

» Finite Difference Approximation to Derivative

A small side note:

* Recall $f(x) \approx f(x') + \frac{\partial f}{\partial x_1}(x')(x - x')$
* Selecting $x = x' + \delta$: $f(x) \approx f(x') + \frac{\partial f}{\partial x_1}(x')\delta$ so $\frac{\partial f}{\partial x_1}(x') \approx \frac{f(x'+\delta) - f(x')}{\delta}$
* Selecting $x = x' - \delta$: $f(x) \approx f(x') - \frac{\partial f}{\partial x_1}(x')\delta$ so $\frac{\partial f}{\partial x_1}(x') \approx \frac{f(x') - f(x'-\delta)}{\delta}$
* Add these together:

$$2\frac{\partial f}{\partial x_1}(x') \approx \frac{f(x'+\delta) - f(x')}{\delta} + \frac{f(x') - f(x'-\delta)}{\delta} = \frac{f(x'+\delta) - f(x'-\delta))}{\delta}$$

i.e.

$$\frac{\partial f}{\partial x_1}(x') \approx \frac{f(x'+\delta) - f(x'-\delta))}{2\delta}$$

$\rightarrow$ an alternative finite difference approximation to the derivative (the *central difference approximation*). Behaves much the same as the $\frac{f(x'+\delta) - f(x'))}{\delta}$ *forward difference* approximation, which to use is largely a matter of taste

» Nesterov Random Search[1]

* Finite difference approach requires $2n$ function evaluations at each iteration. Can we just make *two* function evaluations?

$x = x0$

for k in range(num_iters):

    select a random vector u

    $step = \alpha \frac{f(x + \delta u) - f(x)}{\delta} u$

    $x = x - step$

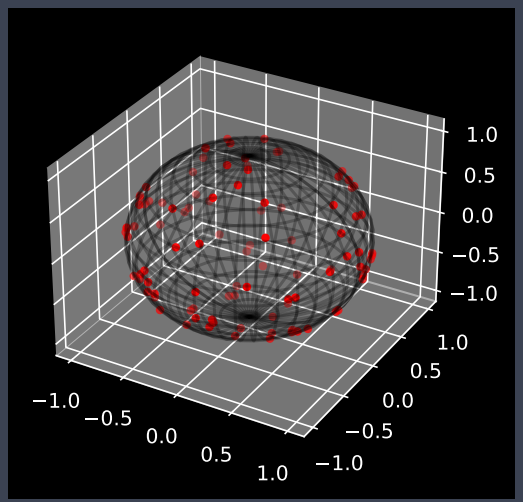
* u is the direction to search in
* $\frac{f(x+\delta u)-f(x)}{\delta}$ is the finite difference approx to the derivative in direction u, use small δ
* α is the step size

[1] Random Gradient-Free Minimization of Convex Functions, 2017. https://link.springer.com/article/10.1007/s10208-015-9296-2

» Nesterov Random Search

- How to select u?
- Select u uniformly at random from the surface of the unit sphere[2]
- Generate a vector z with each element Gaussian distributed. Set new point $u = z/\sum_{i=1}^{n} z_i^2$. In python:

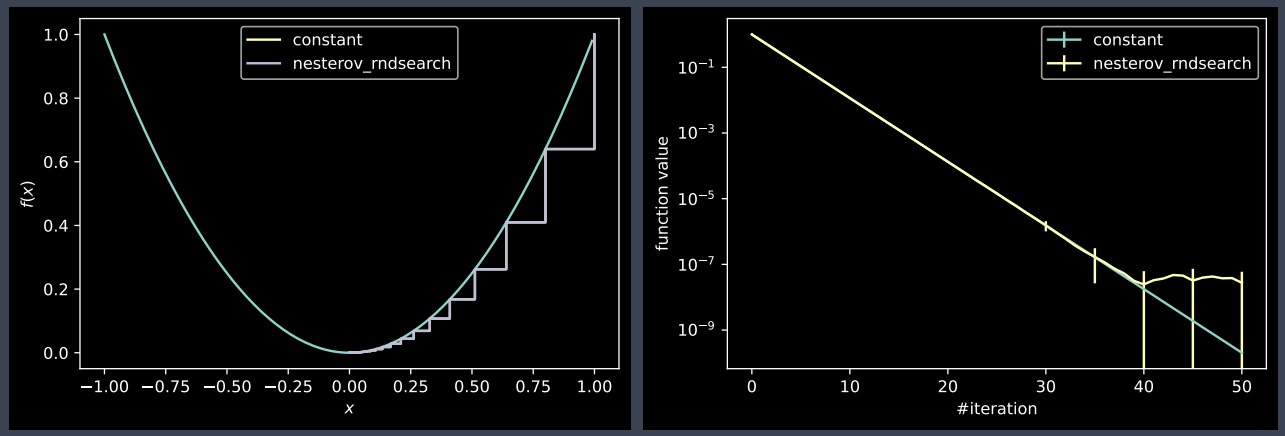
$$z = np.random.randn(ndim)$$
$$u/ = np.sqrt(np.sum(z * z, axis = 0))$$



[2] See https://en.wikipedia.org/wiki/N-sphere#Generating_random_points

» Examples

* $f = x^2$, starting point $x = 1$, grad descent with constant stepsize $\alpha = 0.1$, random search $\alpha = 0.1/\delta = 0.001$

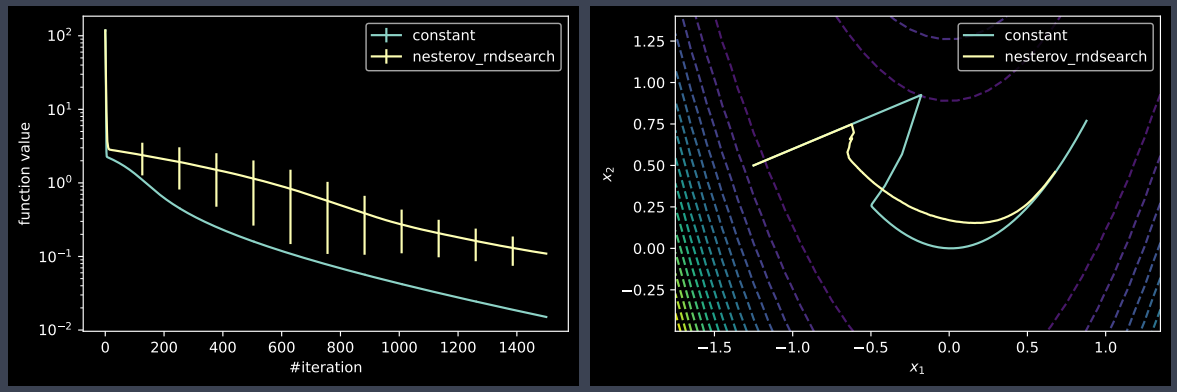


rnd search data is mean and std dev over 25 runs

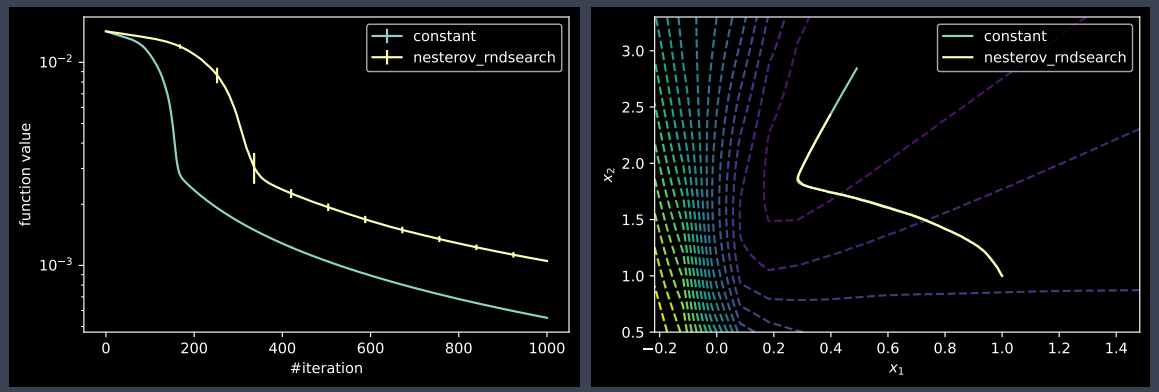
* Nesterov convergence rate is v similar to gradient descent
* Nesterov convergence stops around iteration 40 $\rightarrow$ need to reduce δ

» Examples

* Rosenbrock function: grad descent with constant stepsize $\alpha = 0.002$, random search $\alpha = 0.002/\delta = 0.001$



* Toy neural net loss: grad descent with constant stepsize $\alpha = 0.75$, random search $\alpha = 0.75/\delta = 0.001$



rnd search data is mean and std dev over 25 runs

* *No free lunch principle*: we can expect that generally convergence of Nesterov random search will be around n times slower than grad descent $\rightarrow 2n$ function evals per iteration for finite diff but only 2 function evals for Nesterov
* We roughly see that in above plots (remember $\log 2x = \log 2 + \log x$ so scaling by 2 corresponds to a vertical shift in these plots)