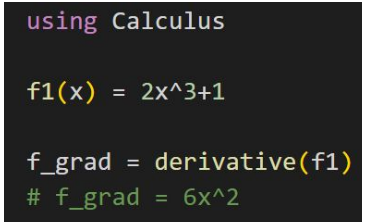
Gradient Descent in julia

“Walks like Python, runs like C”

Jacob Barkovitch

Julia's Native Math Notation

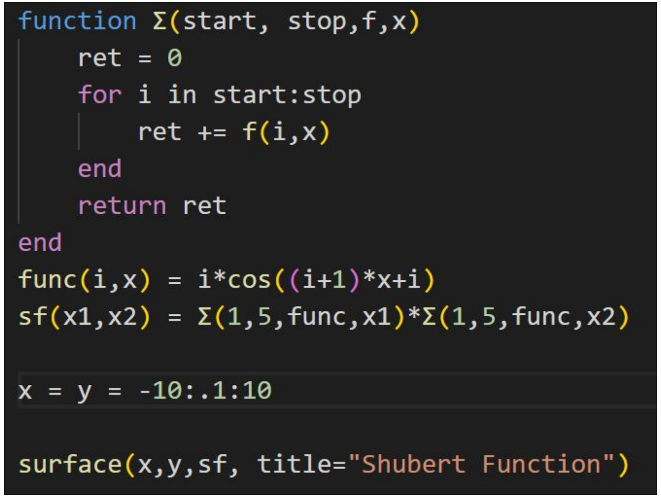


```
using Calculus

f1(x) = 2x^3+1

f_grad = derivative(f1)
# f_grad = 6x^2
```

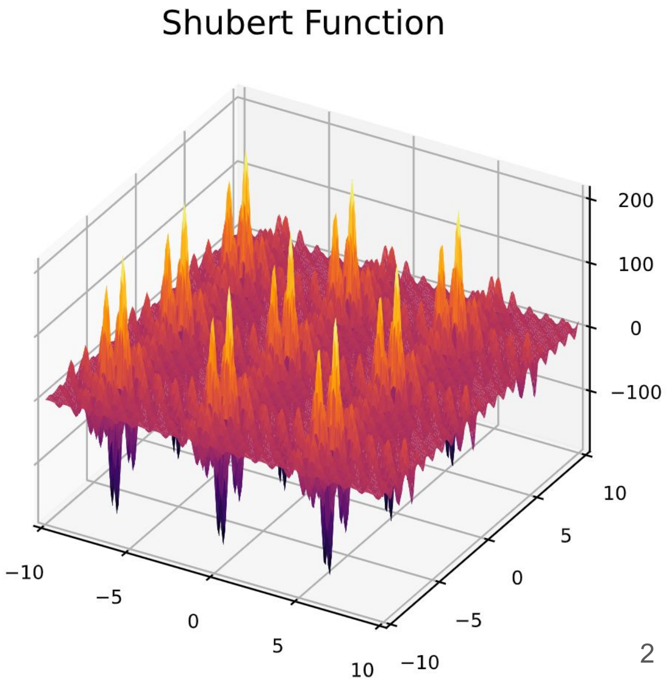
- Julia excels at representing mathematical functions:
  - Shubert function, three hump camel function, derivatives

$$f(x_1, x_2) = \left(\sum_{i=1}^{5} icos[(i+1)x_1 + i]\right)\left(\sum_{i=1}^{5} icos[(i+1)x_2 + i]\right)$$

$$\text{where,} -10 \leq x_i \leq 10 \quad i = 1, 2$$



```
function Σ(start, stop,f,x)
    ret = 0
    for i in start:stop
        ret += f(i,x)
    end
    return ret
end
func(i,x) = i*cos((i+1)*x+i)
sf(x1,x2) = Σ(1,5,func,x1)*Σ(1,5,func,x2)

x = y = -10:.1:10

surface(x,y,sf, title="Shubert Function")
```

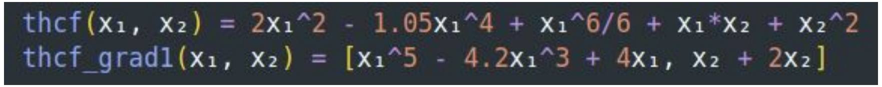


Julia vs. Python Code

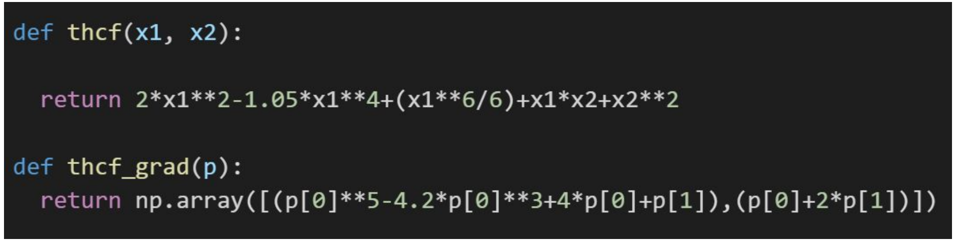
Three Hump Camel Function:

$$f(x_1, x_2) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1 x_2 + x_2^2$$

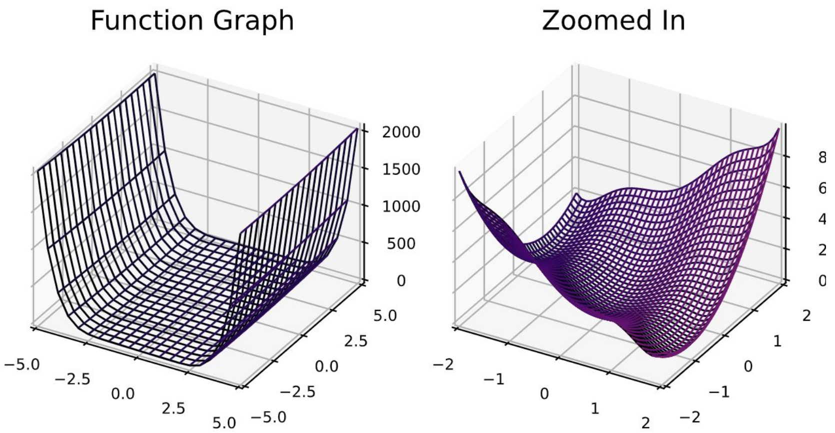
$$\text{where}, -5 \leq x_i \leq 5 \quad i = 1, 2$$



```
thcf(x₁, x₂) = 2x₁^2 - 1.05x₁^4 + x₁^6/6 + x₁*x₂ + x₂^2
thcf_grad1(x₁, x₂) = [x₁^5 - 4.2x₁^3 + 4x₁, x₂ + 2x₂]
```

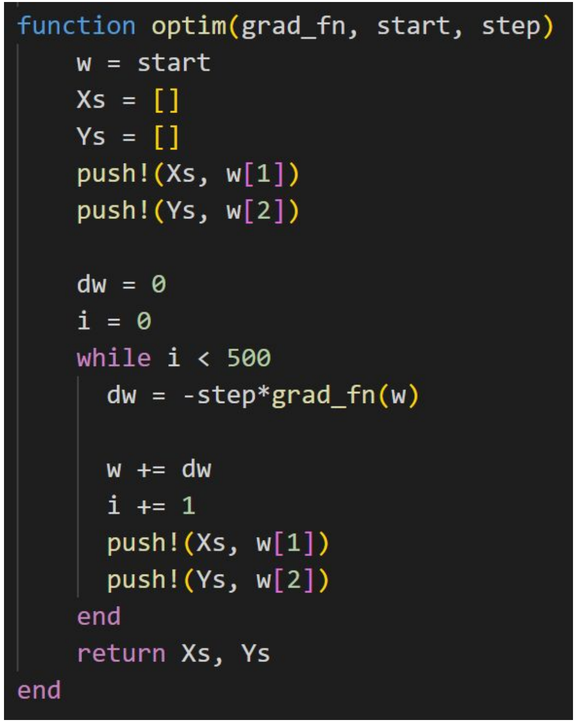


```
def thcf(x1, x2):

  return 2*x1**2-1.05*x1**4+(x1**6/6)+x1*x2+x2**2

def thcf_grad(p):
  return np.array([(p[0]**5-4.2*p[0]**3+4*p[0]+p[1]),(p[0]+2*p[1])])
```


Function Graph

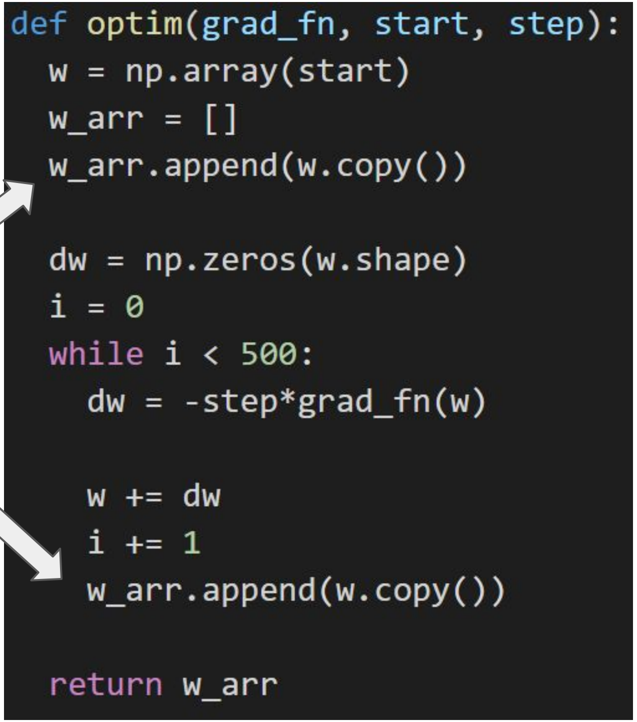
Zoomed In

Gradient Descent Optimizer in Julia & Python



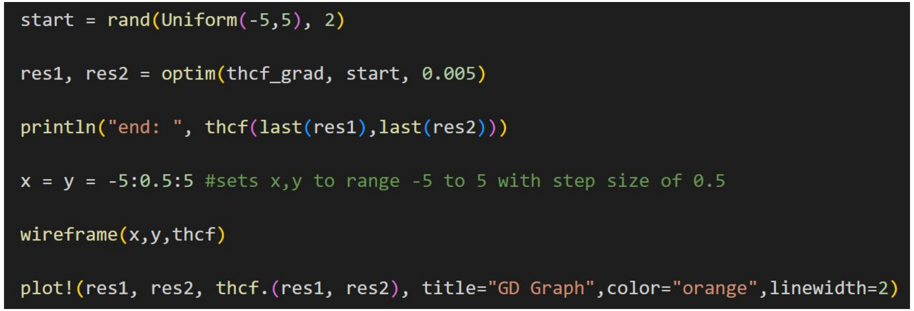
```
function optim(grad_fn, start, step)
    w = start
    Xs = []
    Ys = []
    push!(Xs, w[1])
    push!(Ys, w[2])

    dw = 0
    i = 0
    while i < 500
        dw = -step*grad_fn(w)

        w += dw
        i += 1
        push!(Xs, w[1])
        push!(Ys, w[2])
    end
    return Xs, Ys
end
```

Explicit deep copies in python



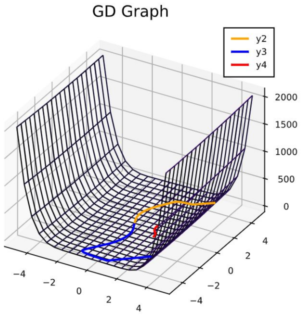
```
def optim(grad_fn, start, step):
  w = np.array(start)
  w_arr = []
  w_arr.append(w.copy())

  dw = np.zeros(w.shape)
  i = 0
  while i < 500:
    dw = -step*grad_fn(w)

    w += dw
    i += 1
    w_arr.append(w.copy())

  return w_arr
```

Running GD and Graphing Results

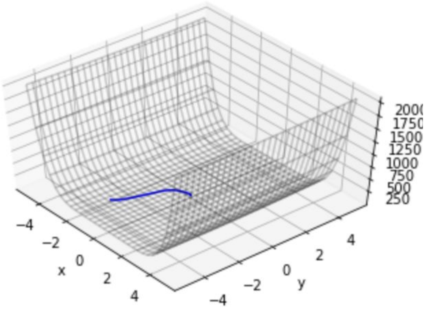


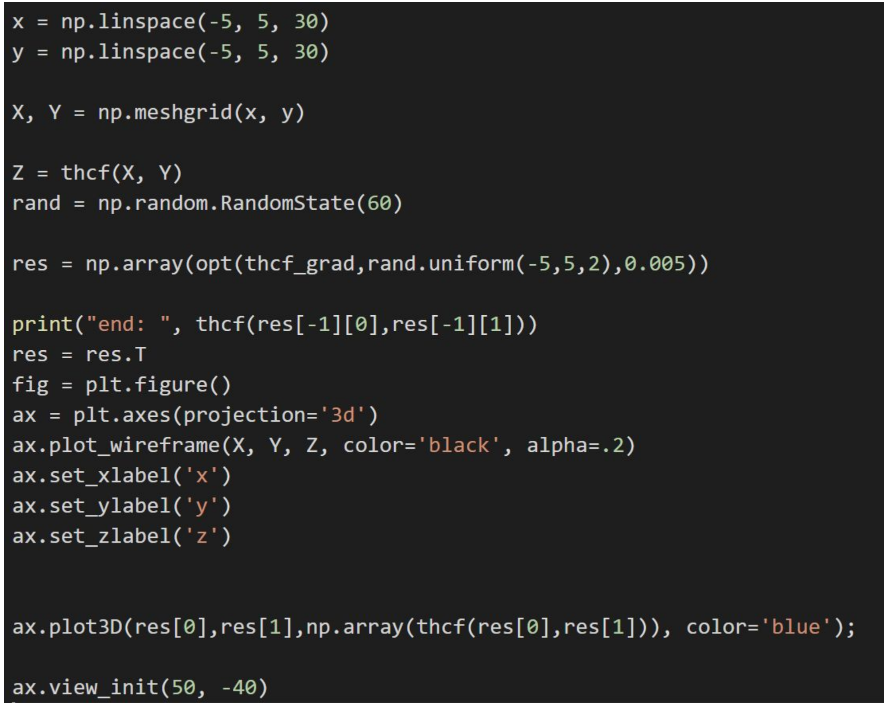
```
start = rand(Uniform(-5,5), 2)

res1, res2 = optim(thcf_grad, start, 0.005)

println("end: ", thcf(last(res1),last(res2)))

x = y = -5:0.5:5 #sets x,y to range -5 to 5 with step size of 0.5

wireframe(x,y,thcf)

plot!(res1, res2, thcf.(res1, res2), title="GD Graph",color="orange",linewidth=2)
```

Julia



Python





```
x = np.linspace(-5, 5, 30)
y = np.linspace(-5, 5, 30)

X, Y = np.meshgrid(x, y)

Z = thcf(X, Y)
rand = np.random.RandomState(60)

res = np.array(opt(thcf_grad,rand.uniform(-5,5,2),0.005))

print("end: ", thcf(res[-1][0],res[-1][1]))
res = res.T
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black', alpha=.2)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')


ax.plot3D(res[0],res[1],np.array(thcf(res[0],res[1])), color='blue');

ax.view_init(50, -40)
```