Christopher Lum
lum@uw.edu

Lecture 10e
Numerical Algorithms for Unconstrained Optimization: Diminishing Step Size

Outline

-Introduction
-Choosing a Step Size
 -Constant Step Size
 -Diminishing Step Size

Introduction

Recall that we are examining algorithms of the form

$$x^{k+1} = x^k + \alpha^k d^k \qquad k = 0, 1, \ldots \qquad \textbf{\textit{(Eq.1.5)}}$$

where α^k = step size (positive scalar)
 d^k = direction (vector)

Recall that for our analysis purposes, we consider d^k to be a unit vector.

We now investigate two simple methods for choosing the step size

1. Constant step size
2. Diminishing step size

Constant Step Size

The simplest technique for choosing a step size is to simply use a constant step size.

$$\alpha^k = \beta \qquad \forall k$$

where β = positive constant

Scenario01: Cost Function f_A, constant step size with $\beta = 0.5$

Consider a cost function of the form

$$f_A(x) = \frac{1}{2} x^T H_A x + g_A{}^T x + r_A$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad H_A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \quad g_A = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad r_A = 2$$

$$f_A(x) = \frac{1}{2}\, x^T H_A\, x + g_A{}^T\, x + r_A$$

where $\quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad H_A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \quad g_A = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad r_A = 2$

In[•]:=

```
x = (x1
     x2); HA = (1 1
                1 2); gA = (1
                            2); rA = 2;
temp = 1/2 Transpose[x].HA.x + Transpose[gA].x + rA;
Print["f_A"]
fA[x1_, x2_] = temp[[1, 1]] // Expand
Print[" "]

Print["∇f_A"]
gradFA[x1_, x2_] = (D[fA[x1, x2], x1]
                    D[fA[x1, x2], x2]) // Simplify;
gradFA[x1, x2] // MatrixForm
Print[" "]

(*Analytically calculate stationary point*)
Print["Stationary point"]
temp = Solve[{gradFA[x1, x2][[1, 1]] == 0, gradFA[x1, x2][[2, 1]] == 0}, {x1, x2}];
x1star = x1 /. temp[[1]]
x2star = x2 /. temp[[1]]
Print[" "]

Print["f(x*)"]
fA[x1star, x2star]
Print[" "]

(*Plot the scenario*)
x1min = -15; x1max = 15; x2min = -15; x2max = 15;
fighContour = ContourPlot[fA[x1, x2], {x1, x1min, x1max},
  {x2, x2min, x2max}, PlotPoints → 25, ColorFunction → Hue]
```

f_A

Out[•]= $2 + x1 + \frac{x1^2}{2} + 2\,x2 + x1\,x2 + x2^2$

∇f_A

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 + x1 + x2 \\ 2 + x1 + 2\,x2 \end{pmatrix}$$

Stationary point

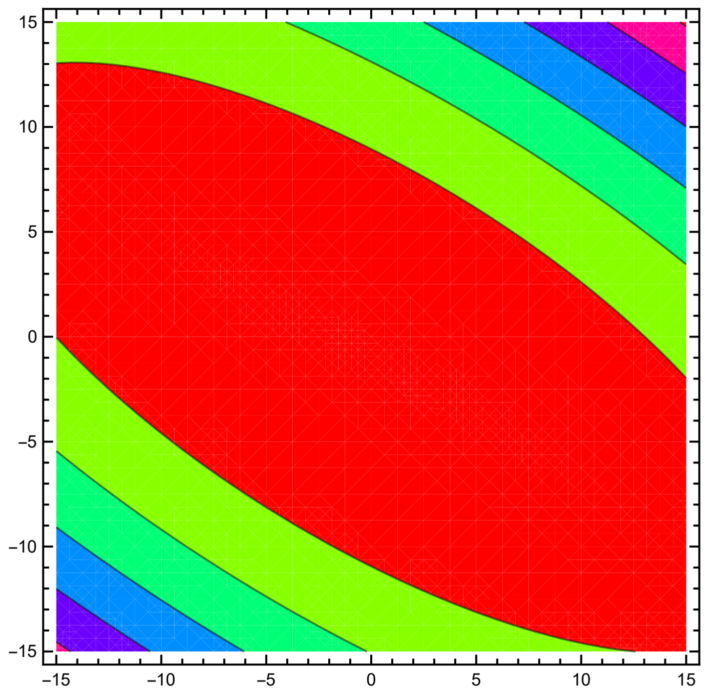
Out[ ]= 0

Out[ ]= -1

$f(x^*)$

Out[ ]= 1

Out[ ]=

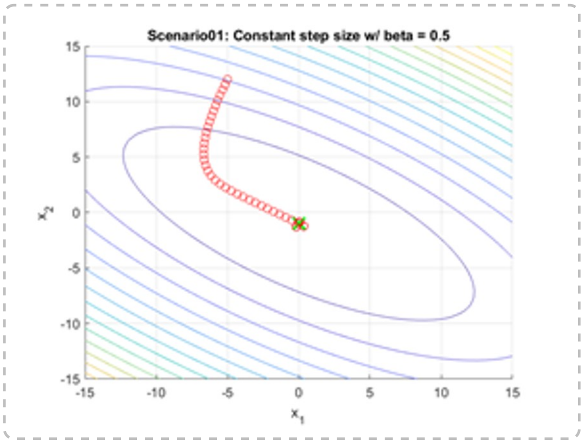


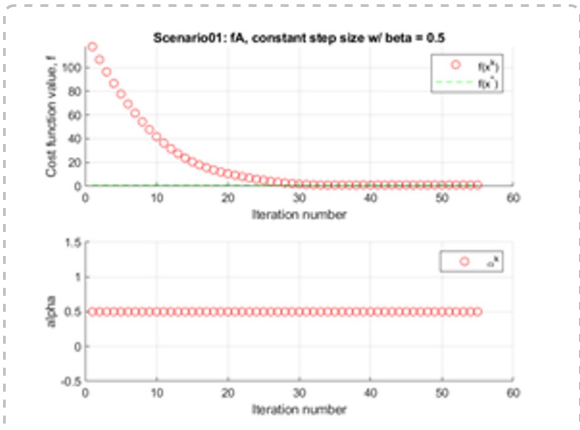
Depending on the initial condition, the algorithm would walk down the contour in successive steps. For example, consider a scenario with

$$\beta = 0.5$$

$$x^0 = \begin{pmatrix} -5 \\ 12 \end{pmatrix}$$

The results are shown below

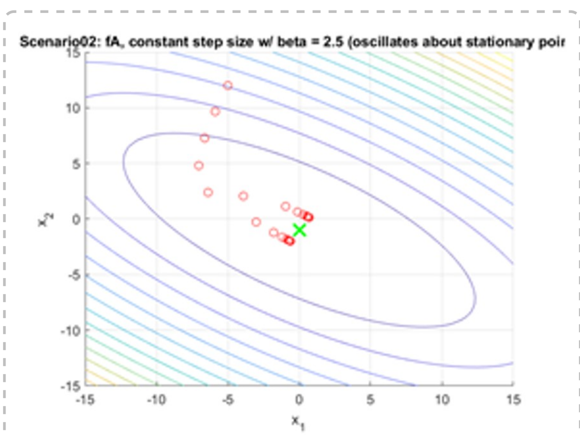


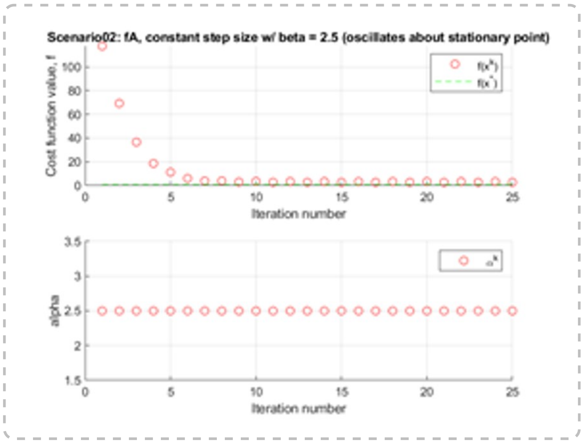


As can be seen, the converge is reasonable but it takes a relatively large number of iterations to reach the stationary point.

Scenario02: Cost Function f_A, constant step size with $\beta = 2.5$

If we increase the step size to $\beta = 2.5$, some interesting behavior emerges





The algorithm “bounces around” the statinoary point as the step size it too large.

	pro	**cons**
smaller step size	able to find stationary point	slow progress
larger step size	fast progress	may miss stationary point

Scenario03: Cost Function f_B, constant step size with $\beta = 2.5$

We can modify the cost function slightly to illustrate another shortcoming of this approach

Consider a cost function of the form

$$f_B(x) = \frac{1}{2} x^T H_B x + g_B{}^T x + r_B$$

where $\quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad H_B = \begin{pmatrix} 1 & 1 \\ 1 & 200 \end{pmatrix} g_B = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad r_B = 101$

In[•]:=
```
HB = (1  1
      1  200); gB = (1
                     200); rB = 101;
temp = 1/2 Transpose[x].HB.x + Transpose[gB].x + rB;
Print["f_B"]
fB[x1_, x2_] = temp[[1, 1]] // Expand
Print[" "]

Print["∇f_B"]
gradFB[x1_, x2_] = (D[fB[x1, x2], x1]
                    D[fB[x1, x2], x2]) // Simplify;
gradFB[x1, x2] // MatrixForm
Print[" "]

(*Analytically calculate stationary point*)
Print["Stationary point"]
temp = Solve[{gradFB[x1, x2][[1, 1]] == 0, gradFB[x1, x2][[2, 1]] == 0}, {x1, x2}];
x1star = x1 /. temp[[1]]
x2star = x2 /. temp[[1]]
Print[" "]

Print["f(x*)"]
fB[x1star, x2star]
Print[" "]

(*Plot the scenario*)
x1min = -15; x1max = 15; x2min = -15; x2max = 15;
fighContour = ContourPlot[fB[x1, x2], {x1, x1min, x1max},
  {x2, x2min, x2max}, PlotPoints → 25, ColorFunction → Hue]
```

f_B

Out[•]= $101 + x1 + \frac{x1^2}{2} + 200\, x2 + x1\, x2 + 100\, x2^2$

∇f_B

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 + x1 + x2 \\ x1 + 200 \times (1 + x2) \end{pmatrix}$$

Stationary point

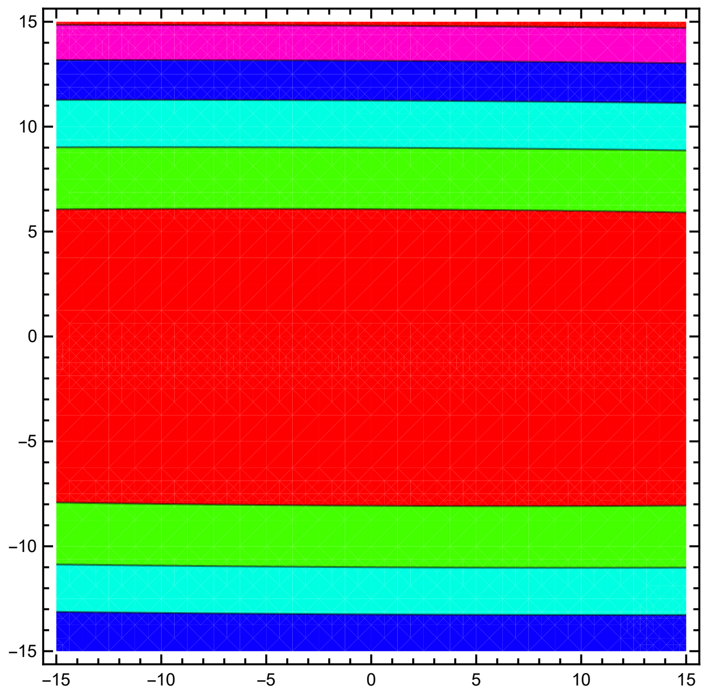
Out[•]= 0

Out[•]= -1

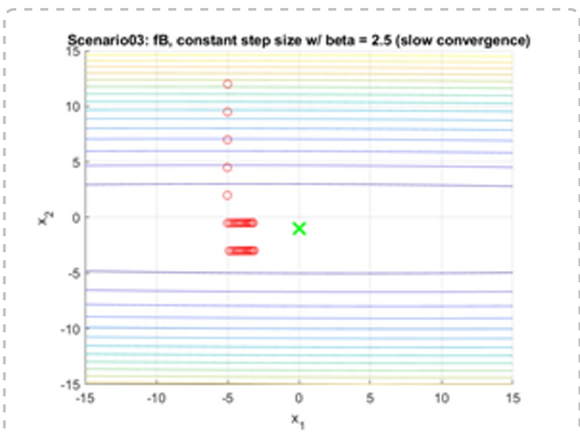
f(x*)

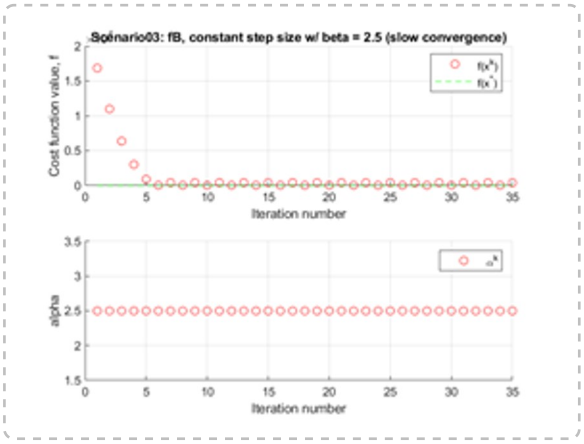
Out[ ]= 1

Out[ ]=



The results are shown below





As can be seen, it makes very slow converge towards to the stationary point due to the poor scaling of the decision variables.

Diminishing Step Size

An alternative is to choose the step size to be proportional to the norm of the gradient.

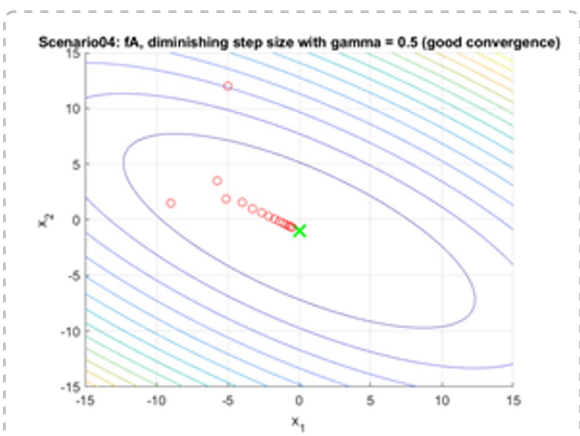
$$\alpha^k = \gamma \, || \nabla f(x^k) ||$$

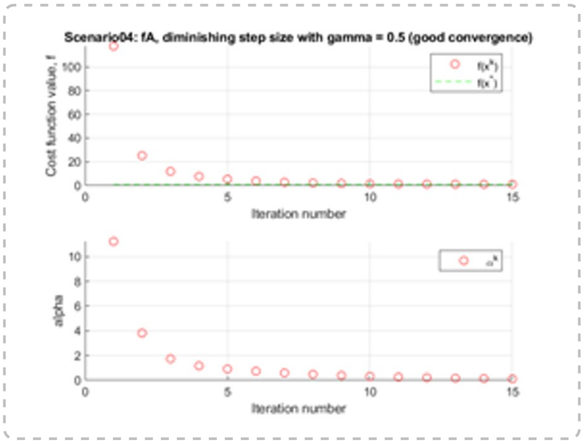
where γ = scaling factor

$|| \nabla f(x^k) ||$ = norm of the gradient at x^k

The behavior is that when the gradient is large, the step size is large and vice versa.

Scenario04: Cost Function f_A, diminishing step size with $\gamma = 0.5$

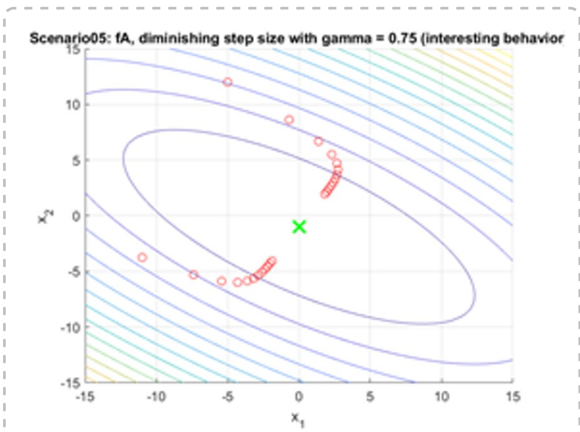


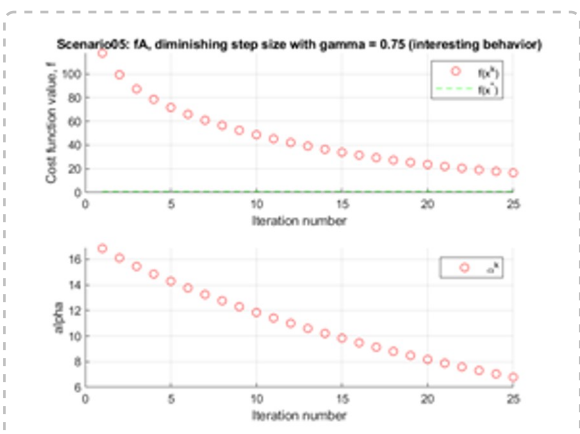


As can be seen, converge is reasonable with initial large steps which then diminish as we approach the stationary point.

Scenario05: Cost Function f_A, diminishing step size with γ = 0.75

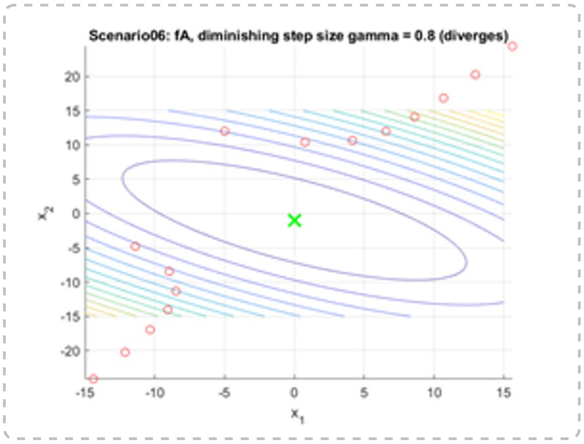
If the parameter γ is not chosen appropriately, interesting/undesirable behavior can emerge. For example, increasing γ to

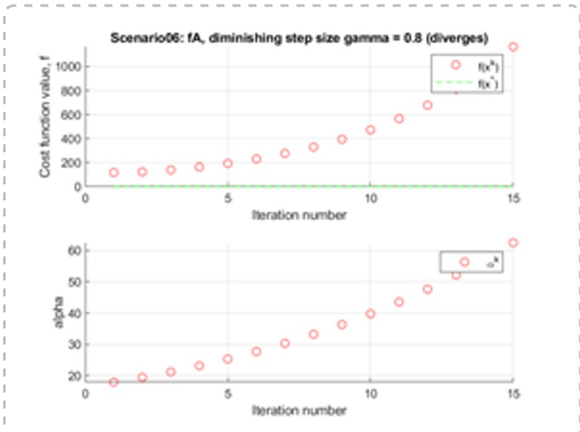




Scenario06: Cost Function f_A, diminishing step size with γ = 0.8

Further increasing γ to 0.8 yields

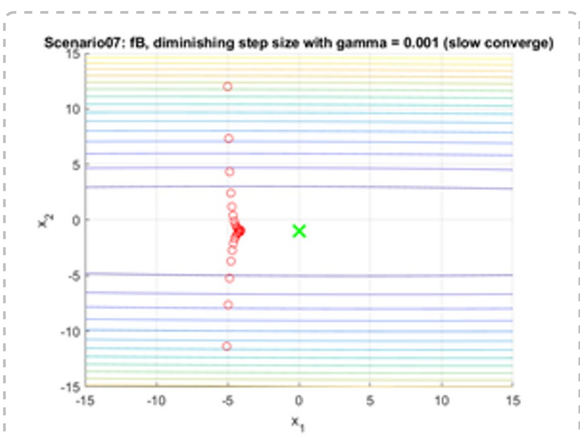


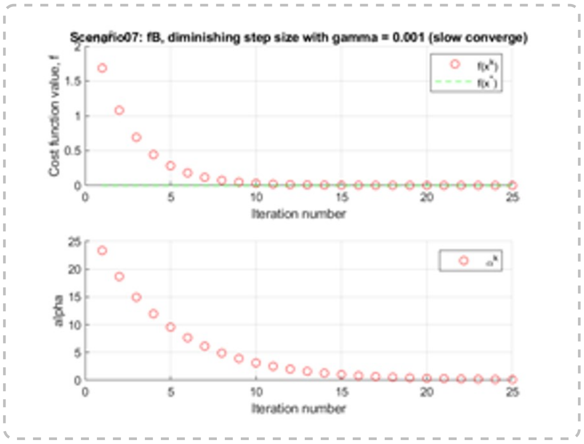


The system diverges as the step size is too large.

Scenario07: Cost Function f_B, diminishing step size with $\gamma = 0.009$

As we saw previously, we need to use a small γ to ensure that the system does not diverge. This problem is exacerbated by examining f_B. If we use $\gamma = 0.01$ the system diverges. As such, we chose $\gamma = 0.009$ which yields the result shown below





So we see that there are still issues with the diminishing step size scheme with the poorly scaled problem.