CMR INSTITUTE OF TECHNOLOGY	USN										

Sub:	Internal Assessment Test III Aug 2024 Optimization Techniques						Code:	BCS405C
Date:	02/08/2024	Duration:	90 mins	Max Marks:	50	Sem: IV	Branch:	**CSDS/CSML**

Answer any five of the following.		Marks	OBE CO	OBE RBT
1	Use Steepest Descent method for $f(x,y) = x - y + 2x^2 + 2xy + y^2$ starting from (0,0)	10	CO3	L1,L2
2	Use Newton Raphson method to find the smallest root and the second smallest positive roots of the equation tan x = x correct to 4 decimal places.	10	CO4	L2,L3
3	Define Hessian matrix. Using Hessian matrix classify the relative extreme for the function $f(x,y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$	2+8	CO3	L3
4	Find the linear regression coefficients using Gradient Descent method.	10	CO4	L3
5	Explain in brief 1. Adagrad optimization strategy 2. RMS prop	5+5	CO5	L2
6	What is the difference between convex optimization and non-convex optimization. Write short notes on ADAM.	5+5	CO5	L2,L3

CMR INSTITUTE OF TECHNOLOGY	USN										

Sub:	Internal Assessment Test III Aug 2024 Optimization Techniques						Code:	BCS405C
Date:	02/08/2024	Duration:	90 mins	Max Marks:	50	Sem: IV	Branch:	**CSDS/CSML**

Answer any five of the following.		Marks	OBE CO	OBE RBT
1	Use Steepest Descent method for $f(x,y) = x - y + 2x^2 + 2xy + y^2$ starting from (0,0)	10	CO3	L1,L2
2	Use Newton Raphson method to find the smallest root and the second smallest positive roots of the equation tan x = x correct to 4 decimal places.	10	CO4	L2,L3
3	Define Hessian matrix. Using Hessian matrix classify the relative extreme for the function $f(x,y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$	2+8	CO3	L3
4	Find the linear regression coefficients using Gradient Descent method.	10	CO4	L3
5	Explain in brief 1. Adagrad optimization strategy 2. RMS prop	5+5	CO5	L2
6	What is the difference between convex optimization and non-convex optimization. Write short notes on ADAM.	5+5	CO5	L2,L3

Working Procedure to Fit a Linear Regression Line to Data Using the Gradient Descent Method

(1) The following steps are used to fit a linear regression line $\hat{y} = a + bx$ for the given data:

Step 1: Define the Model and Cost Function

The linear regression model you want to fit is given by:

$$\hat{y} = a + bx$$

where $\hat{y}$ is the predicted value, a and b are the parameters (weights) of the model and x is the input feature.

The cost function (error function) for linear regression is the Mean Squared Error (MSE):

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y})^2$$

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - (a + bx_i))^2$$

Where n is the number of data points.

Step 2: Initialize Weights and Hyperparameters

Initialize the weights a and b to some arbitrary values. Start with $a = 0$ and $b = 0$. Also need to set hyperparameters:

- Learning rate (α): A small positive value that controls the step size in each iteration.
- Number of iterations: The number of times we update the weights.

Professor, Department of Artificial Intelligence & Data Science,
Don Bosco Institute of Technology, Bangalore.

Step 3: Gradient Descent

For each iteration, calculate the gradients of the cost function with respect to the weights (*a* and *b*) and update the weights accordingly.

The gradients are given by:

$$\Delta a = -\frac{2}{n}\sum_{i=1}^{n}\left(y_i - (a + bx_i)\right)$$

$$\Delta b = -\frac{2}{n}\sum_{i=1}^{n} x_i \times \left(y_i - (a + bx_i)\right)$$

Update the weights using the gradients:

$$a_{new} = a - \alpha \times \Delta a$$

$$b_{new} = b - \alpha \times \Delta b$$

Repeat this process for the specified number of iterations.

Step 4: Predict

After training, use the final values of *a* and *b* to make predictions:

$$\hat{y} = a + bx$$

Step 5: Evaluate and visualize

Evaluate the quality of the linear regression model by calculating the final MSE on your training data:

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n}(y_i - \hat{y})^2$$

Also, visualize the linear regression line by plotting it alongside the data points.

Professor, Department of Artificial Intelligence & Data Science,
Don Bosco Institute of Technology, Bangalore.

Step 6: Iterate as Needed

We need to adjust the learning rate and the number of iterations to find the best-fitting line. If the cost is not converging or fluctuating, you may need to modify the hyperparameters.

This process allows you to iteratively update the weights to minimize the cost function, resulting in a linear regression line that best fits the given data.

Examples

1. We have recorded the weekly average price of a stock over 6 consecutive days. Y shows the weekly average price of the stock and x shows the number of the days. Try to fit the best possible function ' f ' to establish the relationship between the number of the day and conversion rate. (Applying Gradient descent) where f(x) = y = a + b * x.

X	1	2	3	4	5	6
Y	10	14	18	22	25	33

The initial values of a & b are a = 4.9 & b = 4.401. The learning rate is mentioned as .05. The error rate of a & b should be less than .01. Plot the predicted and actual data in a graph.

Solution:

Given data:

X = x_i = 1, 2, 3, 4, 5, 6

Y = y_i = 10, 14, 18, 22, 25, 33

n = 6

Initialization: a = 4. 9 and b = 4.401

Learning rate α = 0.05

Maximum allowable error for a and b = 0.01

The goal is to minimize the MSE (mean squared error) defined as

Batch Gradient Descent	Stochastic Gradient Descent
No random shuffling of points are required.	The data sample should be in a random order, and this is why we want to shuffle the training set for every epoch.
Can't escape shallow local minima easily.	SGD can escape shallow local minima more easily.
Convergence is slow.	Reaches the convergence much faster.
It updates the model parameters only after processing the entire training set.	It updates the parameters after each individual data point.
The learning rate is fixed and cannot be changed during training.	The learning rate can be adjusted dynamically.
It typically converges to the global minimum for convex loss functions.	It may converge to a local minimum or saddle point.
It may suffer from overfitting if the model is too complex for the dataset.	It can help reduce overfitting by updating the model parameters more frequently.

Q9b What is the difference between convex optimization and non convex optimization?

Convex optimization and non-convex optimization are both optimization problems, but they differ in the number of optimal solutions they can have:

- Convex optimization

In convex optimization, there can only be one globally optimal solution, or it may be possible to prove that there is no feasible solution. Convex optimization is easier and more reliable because convex functions have a unique global minimum. Convex problems can also be solved efficiently, even when they are very large. Examples of convex optimization problems include multi-period processor speed scheduling, minimum time optimal control, and grasp force optimization.

- Non-convex optimization

  In non-convex optimization, the objective or some of the constraints are non-convex, which can lead to multiple feasible regions and multiple locally optimal points within each region. This can make optimization more challenging. Non-convex optimization can still be a good choice if the optimization scheme doesn't get stuck in a local minimum. It can also be used to implement more accurate state dynamics. However, even simple-looking non-convex optimization problems with only ten variables can be very challenging, and problems with hundreds of variables can be intractable.

called bias correction, we obtain the corrected first and second impulses respectively.

These correction cause the values of the first and second impulse to be higher at the beginning of the training than without this correction. As a result, the first update step of the neural network weight parameters does not become too large. Thus, the training is not already messed up at the very beginning.

With the additional bias corrections, we obtain the complete form of the ADAM optimizer.

9a)AdaGrad optimization strategy

Another optimization strategy I would like to introduce is called AdaGrad. The idea behind AdaGrad is that you keep a running sum of squared gradients during optimization. In this case, we don't have a momentum term, but an expression , which is **the sum of squared gradients** up to the time .

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

When we optimize a weights θ_j , we divide the current gradient $\nabla_j L$ by the root of the term g_{t+1}. To understand the intuition behind AdaGrad, please imagine a loss function in a two-dimensional space. In this space, the gradient of the loss function **increases very weakly in one direction** and **very strongly in the other direction**. If we now sum up the gradients along the axis in which the gradients increase weakly, the squared sum of these gradients becomes even smaller.

If during the update step we divide the current gradient $\nabla_j L$ by a very small sum of the squared gradients g_{t+1}, the quotient becomes very high. For the other axis, along which the gradients increase sharply, exactly the opposite is true. This means that we speed up the updating process **along the axis with weak gradients** by increasing these gradients along this axis. On the other hand, we slow down the updates of the weights **along the axis with large gradients**.

Disadvantages: there is a problem with this optimization algorithm.

If the training takes too long. Over time, this term the sum of squared gradients would **grow larger.** When the current gradient is divided by this large number, the update step for the weights becomes very small. It is as if we were using **a very low learning rate**, which becomes even lower the longer the training takes. In the worst case, we would get stuck at AdaGrad and the training would go on forever.

9a) &10c)RMSProp

There is a slight modification of AdaGrad called "RMSProp". This modification is intended to solve the previously described problem that can occur with AdaGrad. In RMSProp, the running sum of squared gradients g_{t+1} is maintained. However, instead of allowing this sum to increase continuously over the training period, we allow the sum to decrease.

AdaGrad	RMS Prop
$g_0 = 0$	$g_0 = 0, \alpha \simeq 0.9$
$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$	$g_{t+1} \leftarrow \alpha \cdot g_t + (1-\alpha)\nabla_\theta \mathcal{L}(\theta)^2$
$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$	$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$

For RMSProp, the sum of squared gradients is multiplied by a decay rate α and the current gradient – weighted by (1- α) – is added. The update step in the case of RMSProp looks the same as in AdaGrad. Here we divide the current gradient by the sum of the squared gradients to get the nice property

of speeding up the updating of the weights along one dimension and slowing down the motion along the other.

Although SGD with momentum is able to find the global minimum faster, this algorithm takes a much longer path that could be dangerous. This is because a longer path means **more potential saddle points and local minima** of the loss function that could lie along that path. RMSProp, on the other hand, goes straight to the global minimum of the loss function without taking a detour.

1. **Handling Non-stationary Objectives**:
    - **RMSProp** is particularly well-suited for non-stationary objectives (where the data distribution changes over time), as it can adjust more dynamically to the changes compared to Adagrad.
2. **Empirical Performance**:
    - In practice, RMSProp often performs better than Adagrad on a variety of machine learning tasks. It tends to converge faster and reach better solutions, especially when dealing with deep learning models.

Overall, RMSProp is generally preferred for its ability to maintain a more stable and effective learning rate throughout training, leading to better performance on many complex tasks.

9c) Describle the saddle point problem in machine learning.

Key Characteristics of a Saddle Point:

1. **Zero Gradient:**

- At a saddle point, the gradient of the cost function is zero. This means that the partial derivatives with respect to each parameter are all equal to zero.

2.**Neither Minimum nor Maximum:**

10a)Stochastic Gradient Descent with Momentum

The first of the four algorithms I would like to introduce is called “Stochastic Gradient Descent with Momentum”:

SGD

$$\theta_j \leftarrow \theta_j - \epsilon \nabla_{\theta_j} \mathcal{L}(\theta)$$

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

GL. 2 Stochastic GD (left), SGD with momentum (right).

On the left side in GL. 2 is the formula for the weight updates according to the regular stochastic gradient descent (SGD for short). The equation on the right represents the rule for the updates of the weights according to the SGD with momentum. **Momentum appears here as an additional term** , which is added to the regular update rule.

Intuitively speaking, by adding this impulse term, we let our **gradient build up some sort of velocity** V during training. The velocity is the running sum of the gradients weighted by ρ.

The parameter ρ can be thought of as friction that “slows” the velocity down a bit. In general, velocity can be seen to increase with time. By using the momentum term, **saddle points and local minima become less dangerous** for the gradient. This is because the step size toward the global minimum now depends not only on the slope of the loss function at the current point, but **also on the velocity** that has built up over time.

For a physical representation of stochastic gradient descent with momentum, imagine a ball rolling down a hill, increasing in velocity with time. If this ball encounters an obstacle along the way, such as a hole or flat ground with no slope, its built-up velocity v would give the ball enough force to roll over this

obstacle. In this case, the **flat ground represents a saddle point** and the **hole represents a local minima** of a loss function.

Both algorithms try to reach the global minimum of the loss function, which is in a 3D space. Momentum term results in the individual gradients having less variance and thus less zig-zagging.

10a)ii)ADAM

We take the best of Adagrad and RMS prop and combine these ideas into a single algorithm called as ADAM.

The main part of this optimization algorithm consists of the following three equations. These equations may seem complicated at first glance, but if you look closely, you will see some similarities with the last three optimization algorithms.

$$m_0 = 0, v_0 = 0$$

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta)$$ Impuls

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$ RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$ RMS Prop + Impuls

The first expression looks a bit like SGD with momentum. In this case, the term m_t would be the velocity and the term β_1 would be the friction term. In the case of ADAM, we refer to m_t as the "first momentum." On the other hand, β_1 is just a hyperparameter. However, the difference with SGD with momentum is the factor $1-\beta_1$ multiplied by the current gradient.

The second expression **can be considered as RMSProp**, where we keep the running sum of squared gradients. Also in this case, there is the factor $1-\beta_2$, which is multiplied by the squared gradient.

The term v_t in the equation is called the "second momentum" and is also just a hyperparameter. The final update equation can be viewed as **a combination of RMSProp and SGD with momentum**.

Disadvantages

At the very first time step t=0, the first and second pulse terms $m0$ and $v0$ are set to zero. After the first update of the second momentum $v1$, this term is still very close to zero. When we update the weight parameters in the last expression in GL. 5, we divide by a very small second momentum term $v1$. This leads to a very large first update step.

To address the problem of large update steps happening at the beginning of training, ADAM includes a correction clause:

$$m_{t+1} \leftarrow \beta_1 m_t + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta)$$ Impuls

$$v_{t+1} \leftarrow \beta_2 v_t + (1-\beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$ RMS Prop

$$\hat{m}_{t+1} \leftarrow \frac{m_{t+1}}{1-\beta_1^t}$$

$$\hat{v}_{t+1} \leftarrow \frac{v_{t+1}}{1-\beta_2^t}$$ Bias Korrektur

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$ RMS Prop + Impuls

After the initial update of the first and second pulses, we **make an unbiased estimate of these pulses** by considering the current time step. With the so-

called bias correction, we obtain the corrected first and second impulses respectively.

These correction cause the values of the first and second impulse to be higher at the beginning of the training than without this correction. As a result, the first update step of the neural network weight parameters does not become too large. Thus, the training is not already messed up at the very beginning.

With the additional bias corrections, we obtain the complete form of the ADAM optimizer.

9a)AdaGrad optimization strategy

Another optimization strategy I would like to introduce is called AdaGrad. The idea behind AdaGrad is that you keep a running sum of squared gradients during optimization. In this case, we don't have a momentum term, but an expression , which is **the sum of squared gradients** up to the time .

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

When we optimize a weights θ_j , we divide the current gradient $\nabla_j L$ by the root of the term g_{t+1}. To understand the intuition behind AdaGrad, please imagine a loss function in a two-dimensional space. In this space, the gradient of the loss function **increases very weakly in one direction** and **very strongly in the other direction**. If we now sum up the gradients along the axis in which the gradients increase weakly, the squared sum of these gradients becomes even smaller.

Q5 a) A local max/min occurs at $x = c$ when $f(c) > f(x)$ for x values around c.

A global max/min occurs at $x = c$ if $f(x) \leq f(c)$ for all values of x in the domain.

It is possible to have several global max/min if the function reaches its peak value at more than one point.

Q5 b) The Hessian matrix is a square matrix that represents the 2nd order partial derivatives of a scalar valued function. It provides information about the local curvature of the function and is used in optimisation and analysis of functions to determine concavity or convexity.

For a function $f: \mathbb{R}^n \to \mathbb{R}$ that is twice continuously differentiable, the Hessian matrix is defined as

$$H(f) = \begin{bmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 & \cdots & \partial^2 f/\partial x_1 \partial x_n \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 & \cdots & \partial^2 f/\partial x_2 \partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2 f/\partial x_n \partial x_1 & \partial^2 f/\partial x_n \partial x_2 & \cdots & \partial^2 f/\partial x_n^2 \end{bmatrix}$$

(5b) Find the relative extrema of the function

$$f(x,y) = \frac{1}{3}x^3 + xy^2 - 8xy + 3$$

Ans $f_x = x^2 + y^2 - 8y \qquad f_y = 2xy - 8x$

Set $f_x = 0 \qquad f_y = 0$

$x^2 + y^2 - 8y = 0 - (1) \qquad 2xy - 8x = 0 - (2)$

$2x(y-4) = 0$

$\Rightarrow x = 0, \; y = 4$

Case 1 $x = 0$

(1) is $y^2 - 8y = 0 \Rightarrow y(y-8) = 0 \Rightarrow y = 0, 8$

We have 2 critical points $(0,0)$ $(0,8)$

Case 2 $y = 4$ (1) is $x^2 + 16 - 32 = 0 \Rightarrow x = \pm 4$

We have 2 more critical points $(4,4)$ & $(-4,4)$

Step 2 $H = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix} = \begin{pmatrix} 2x & 2y-8 \\ 2y-8 & 2x \end{pmatrix} - (3)$

Evaluate H at critical points

$H(0,0) = \begin{pmatrix} 0 & -8 \\ -8 & 0 \end{pmatrix}$ $\quad Det = -64 < 0$ indicating a saddle point

$H(0,8) = \begin{pmatrix} 0 & 8 \\ 8 & 0 \end{pmatrix}$ $\quad Det = -64$ indicating a saddle point

$H(4,4) = \begin{pmatrix} 8 & 0 \\ 0 & 8 \end{pmatrix}$ $\quad Det = 64$ $\quad f_{xx} = 8 > 0$ indicating a local minimum

$H(-4,4) = \begin{pmatrix} -8 & 0 \\ 0 & -8 \end{pmatrix}$ $\quad Det = 64$ $\quad f_{xx} = -8 < 0$ indicating a local max

Ans $(0,0)$ saddle point

$(0,8)$ saddle point

$(4,4)$ local minimum $\quad (-4,4)$ local maximum

MBP-7a)
Use Steepest Descent Method for $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$ starting from $X_1 = (0,0)$

Ans $\nabla f = \begin{pmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{pmatrix} = \begin{pmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{pmatrix}$

Hessian Matrix $H = \begin{pmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2 \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}$

Iteration 1 Step 1
find S_1 at $X_1 = (0,0)$

$S_1 = -\nabla f(X_1) = -\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Step 2 Compute λ_1 at X_1

$$\lambda_1 = \frac{S_1^T S_1}{S_1^T H S_1} = \frac{\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}}{\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}} = 1$$

New point $X_2 = X_1 + \lambda_1 S_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 1\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Step 3 Check the optimum

$\nabla f(X_2) = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ not optimum

Iteration 2 Step 1 Find S_2 at $X_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

$S_2 = -\nabla f(X_2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Step 2 Compute λ_2 at X_2

$$\lambda_2 = \frac{S_2^T S_2}{S_2^T H S_2} = \frac{(1\ 1)\begin{pmatrix} 1 \\ 1 \end{pmatrix}}{(1\ 1)\begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}} = \frac{1}{5}$$

Hence the new point

$$x_3 = x_2 + \lambda_2 s_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.8 \\ 1.2 \end{pmatrix}$$

Step 3 Check the optimum

$$\nabla f(x_3) = \begin{pmatrix} +0.2 \\ -0.2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$ not an optimum

x_3 is not optimum

Iteration 3 Step 1 $x_3 = \begin{pmatrix} -0.8 \\ 1.2 \end{pmatrix}$

$$s_3 = -\nabla f(x_3) = -\begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix} = \begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix}$$

Step 2 $$\lambda_3 = \frac{s_3^T s_3}{s_3^T H s_3} = \frac{\begin{pmatrix} -0.2 & 0.2 \end{pmatrix}\begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix}}{\begin{pmatrix} -0.2 & 0.2 \end{pmatrix}\begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix}}$$

$$= 1$$

Hence the new point is

$$x_4 = x_3 + \lambda_3 s_3 = \begin{pmatrix} -0.8 \\ 1.2 \end{pmatrix} + 1\begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix}$$

$$= \begin{pmatrix} -1 \\ 1.4 \end{pmatrix}$$

Step 3 Check the optimum

$$\nabla f(x_4) = \begin{pmatrix} -0.2 \\ -0.2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

x_4 is not an optimum

Iteration 4 Step 1 $s_4 = -\nabla f(x_4) = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}$

Step 2 $$\lambda_4 = \frac{s_4^T s_4}{s_4^T H s_4} = \frac{\begin{pmatrix} 0.2 & 0.2 \end{pmatrix}\begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}}{\begin{pmatrix} 0.2 & 0.2 \end{pmatrix}\begin{pmatrix} 4 & 2 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}}$$

$$= 1/5$$

Hence the new point is $x_5 = x_4 + \lambda_4 s_4$

$$= \begin{pmatrix} -1 \\ 1.4 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix} = \begin{pmatrix} -0.96 \\ 1.44 \end{pmatrix}$$

Step 3 $\nabla f(x_5) = \begin{pmatrix} 1 + 4(-0.96) + 2(1.44) \\ -1 + 2(-0.96) + 2(1.44) \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

x_5 is an optimum

$f(x) = x - \tan x$ has a simple root near 4.5

$f'(x) = 1 - \sec^2 x$

$x_0 = 4.5 \qquad x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

$$x_1 = 4.5 - \frac{f(4.5)}{f'(4.5)}$$

$f(4.5) = -0.1373 \qquad f'(4.5) = -21.5048$

$$x_1 = 4.5 - \left(\frac{-0.1373}{-21.5048}\right) = 4.4936$$

$x_0 = 0.1$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$= 0.1 - \frac{f(0.1)}{f'(0.1)} = 0.1 - \frac{(0.10033 - 0.1)}{\sec^2(0.1) - 1}$$

$$0.1 - \frac{(0.10033) - (0.1)}{(0.990025 - 1)}$$

$$= 0.1 - \frac{(0.00033)}{(-0.009975)}$$

$$0.1 + 0.033$$

$$= 0.1033$$

Ans 4.4936, 0.1033