Recitation 16

Tuesday November 4, 2024

1 Recap: Minimization and root finding

Algorithm 1: Gradient Descent

Input: initial guess $\mathbf{x}_0$, step size $\alpha > 0$;
while *not converged* **do**
  $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$
end
return $\mathbf{x}_k$;

For a convex quadratic function $\frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T x$ (with SPD A), gradient descent (GD) (aka "steepest descent") converges when the eigenvalues of $(I - \alpha A)$ have a magnitude less than 1. Let λ_1 be the largest eigenvalue of A and λ_n be the smallest. In the simplest variant where the "learning rate" α is fixed, gradient descent converges when $0 < \alpha < \frac{2}{|\lambda_1|}$.

1. Convergence rate $R = \|I - \alpha A\|$ (= biggest-magnitude eigenvalue since this is symmetric): the residual $r = b - Ax$ decreases by a factor of at least R on each step.

2. The optimal (fastest, smallest R) convergence rate is achieved at $\alpha = \frac{2}{\lambda_1 + \lambda_n}$.

3. The condition number of the matrix A, denoted as $\kappa = \sigma_1/\sigma_n = \lambda_1/\lambda_n$ (for an SPD matrix), influences convergence. Smaller κ indicates more uniform and faster convergence: $R = \frac{\kappa-1}{\kappa+1}$.

1.1 Jacobians

The Jacobian matrix of a vector-valued function $F : \mathbb{R}^n \to \mathbb{R}^m$ is the $m \times n$ matrix of all its first-order partial derivatives: $\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{F}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{F}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial F_1}{\partial x_1} & \cdots & \dfrac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial F_m}{\partial x_1} & \cdots & \dfrac{\partial F_m}{\partial x_n} \end{bmatrix}$ So that it is the linear operator predicting a small change in the "output" for a small change $\delta\mathbf{x} \in \mathbb{R}^n$ in the input:

$$\mathbf{F}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{F}(\mathbf{x}) = J\delta\mathbf{x} + (\text{higher-order terms})\,.$$

An important application of the Jacobian is to perform Newton steps $\mathbf{x} \longrightarrow \mathbf{x} - J^{-1}\mathbf{F}(\mathbf{x})$ to find a root where $\mathbf{F} = 0$. In the special case $\mathbf{F} = \nabla f$, the gradient of a scalar-valued function, then the Jacobian is the **Hessian** matrix $H = H^T$ of *second* derivatives of f: $H_{ij} = \partial^2 f/\partial x_i \partial x_j$. Newton steps $\mathbf{x} \longrightarrow \mathbf{x} - H^{-1}\nabla f$ can therefore be used to accelerate finding a minimum (or extremum) of f: once you are close to a root, Newton iterations *double* the number of digits on every iteration.

2 Exercises

1. Determine the gradients for the following quadratic functions:
   (a) $f(x, y) = 4xy + x^2 + 4y^2$
   (b) $g(x, y) = 2xy + 2x^2 + 2y^2$

2. For the functions $f(x, y)$ and $g(x, y)$ from the previous question, determine whether they are smooth and strongly convex.

3. In this problem, we apply gradient descent to minimize the function $g(x, y)$.

$$g(x, y) = 2xy + 2x^2 + 2y^2$$

   (a) Write the update step in terms of the previous point (x_k, y_k) and step size α.
   (b) Determine an appropriate step size α for the gradient descent algorithm.
   (c) Find the condition number of the matrix associated with $g(x, y)$.
   (d) Find the rate of convergence for the gradient descent applied to $g(x, y)$.
   (e) Implement a few iterations of the gradient descent algorithm, starting from the initial point $(x_0, y_0) = (32, 16)$.
   (f) Employ any relevant methods learned for unconstrained QPs to determine the values of (x^*, y^*) that minimize the function $g(x, y)$. Is the outcome of your gradient descent iterations in part (e) consistent the convergence inequality highlighted in the recap?

4. Jacoians:
   (a) For the vector function $F(\vec{x}) = \begin{bmatrix} f(\vec{x}) \\ g(\vec{x}) \end{bmatrix}$, where $\vec{x} = (x, y)$ and f and g are the scalar-valued functions from problem (1), find the Jacobian matrix.
   (b) For the coordinate transformation given by $F(r, \varphi) = \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix}$, find the Jacobian matrix with respect to the variables r and φ.
   (c) Using the previous answer, would you apply Newton iterations to find the r, φ that solve $F(r, \varphi) = \vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ for some given x, y. *Optional*: The exact solution is $r = \sqrt{x^2 + y^2}$ and $\varphi = \text{atan2}(y, x)$, but it is instructive to try a few Newton iterations in Julia to see how fast it converges. Try $x = y = 1/\sqrt{2}$, which should converge to $r = 1, \varphi = \pi/4$, with an initial guess of $r = 2, \varphi = 0$.

3 Solutions

1. (a) $f(x, y) = w^T Aw$ where $w = \begin{bmatrix} x \\ y \end{bmatrix}$ and $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$.

   Therefore, $\nabla f(x) = 2Aw = \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4y + 2x \\ 4x + 8y \end{bmatrix}$

   (b) Similarly to part (a), $g(x, y) = v^T Bv$ where $v = \begin{bmatrix} x \\ y \end{bmatrix}$ and $B = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

   Therefore, $\nabla g(x) = 2Bv = 2 \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} v = \begin{bmatrix} 4x + 2y \\ 2x + 4y \end{bmatrix}$

2. The Hessian matrices for f and g have the following eigenvalues

$$H_f = 2A = \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} \Rightarrow \text{eigenvalues} = 0 \text{ and } 10$$

$$H_g = 2B = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} \Rightarrow \text{eigenvalues} = 2 \text{ and } 6$$

   Both functions $f(x, y)$ and $g(x, y)$ are smooth and convex since they the eigenvalues for the Hessian matrix are non-negative and bounded by 10 for H_f and by 6 for H_g.

   For strong convexity, the Hessian matrix should be positive definite, which means that only $g(x, y)$ is strongly convex.

3. (a) The update step is

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \alpha \nabla g(x_k, y_k) = \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \alpha 2B \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$
$$= (I - \alpha B) \begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 - 4\alpha & -2\alpha \\ -2\alpha & 1 - 4\alpha \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

   (b) Since g is smooth and strongly convex, we can choose $\alpha = \frac{2}{2+6} = 0.25$.

   (c) The Hessian matrix is $\begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$, and its condition number is the ratio of the largest to the smallest eigenvalue, which is $\frac{6}{2} = 3$.

   (d) The rate of convergence is the largest singular value of $I - 0.25 * 2B$, which is 0.5.

   (e) We apply the following update with $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} 32 \\ 16 \end{bmatrix}$

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & -0.5 \\ -0.5 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

   After 5 iterations, $x_5, y_5 = -0.5, -1.0$.

   (f) There are many ways to find the minimum of $g(x, y)$ since the function is convex. For example, you can solve for the gradients or apply the eigendecomposition technique. One way is to express $g(x, y) = x^2 + y^2 + (x + y)^2$, implying that the minimum is achieved when $(x^*, y^*) = (0, 0)$. We observe that

(x_5, y_5) is close to the true minimum. The convergence inequality for smooth and strongly convex functions is

$$f(x_5, y_5) - f(x^*, y^*) \leq \frac{L}{2}\left(\frac{\kappa-1}{\kappa+1}\right)^{10} \left\| \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \begin{bmatrix} x_* \\ y_* \end{bmatrix} \right\|^2$$

$$\Rightarrow 3.5 \leq \frac{6}{2}\left(\frac{2}{4}\right)^{10} (32^2 + 16^2) = 3.75$$

4. (a) $J_F(x) = \begin{bmatrix} \nabla f(x)^T \\ \nabla g(x)^T \end{bmatrix} = \begin{bmatrix} 2x^T A^T \\ 2x^T B^T \end{bmatrix} = \begin{bmatrix} 4y + 2x & 4x + 8y \\ 2y + 4x & 2x + 4y \end{bmatrix}$

   (b) $J(r, \varphi) = \begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial x}{\partial \varphi} \\ \dfrac{\partial y}{\partial r} & \dfrac{\partial y}{\partial \varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi & -r\sin\varphi \\ \sin\varphi & r\cos\varphi \end{bmatrix}$

   (c) We are trying to find a root of $G(r, \varphi) = F(r, \varphi) - \vec{x}$. Since $\vec{x}$ is a constant, the Jacobian of G is the same as the Jacobian of F, so our Newton iterations are $\begin{pmatrix} r \\ \varphi \end{pmatrix} \longrightarrow \begin{pmatrix} r \\ \varphi \end{pmatrix} - J(r, \varphi)^{-1}(F(r, \varphi) - \vec{x})$. Let's try a few Newton iterations in Julia as suggested. After only 5 iterations, it converges to 8 digits (and would get 16 digits in 6 iterations)!

```
julia> F(r, phi) = [r*cos(phi), r*sin(phi)];

julia> J(r, phi) = [cos(phi) -r*sin(phi); sin(phi) r*cos(phi)];

julia> x = y = 1/sqrt(2)
0.7071067811865475

julia> c = [2, 0] # initial guess as a vector c = [r, phi]
2-element Vector{Int64}:
 2
 0

 julia> c = c - J(c...) \ (F(c...) - [x, y]) # Newton step
2-element Vector{Float64}:
 0.7071067811865475
 0.35355339059327373

julia> c = c - J(c...) \ (F(c...) - [x, y]) # Newton step
2-element Vector{Float64}:
 0.9081951715022664
 0.9454681318524669

julia> c = c - J(c...) \ (F(c...) - [x, y]) # Newton step
2-element Vector{Float64}:
 0.9872161337103353
 0.7699692016739068
```

```
julia> c = c - J(c...) \ (F(c...) - [x, y]) # Newton step
2-element Vector{Float64}:
 0.9998809759312631
 0.7855973392747698

julia> c = c - J(c...) \ (F(c...) - [x, y]) # Newton step
2-element Vector{Float64}:
 0.9999999801644849
 0.78539813968922

julia> c - [1, pi/4] # error compared to exact answer
2-element Vector{Float64}:
 -1.9835515119304148e-8
 -2.3708228269470055e-8
```