Introduction to Optimization (236330) - Exercises Booklet

Spring 2015 - Prof. Michael Zibulevsky

Samuel Londner

Contents

Part I
Multivariate Differentials

1 Gradient of a composed vector function

$f(x) : \mathbb{R}^n \to \mathbb{R}$. Given $\nabla_x f(x)$ (the gradient of $f(x)$ with respect to x), compute the gradient of $f(A \cdot x)$, where A is a matrix.

Solution

We denote $u \triangleq A \cdot x$. Then:

$$\mathrm{d}f(u) = \nabla_u f(u)^T \mathrm{d}u$$

We substitute u for Ax and use the transpose property $A^T B^T = (BA)^T$:

$$\mathrm{d}f(u) = \nabla_u f(Ax)^T \cdot A \cdot dx = \left[A^T \nabla_u f(Ax)\right]^T \mathrm{d}x = \langle A^T \nabla_u f(Ax), \mathrm{d}x\rangle$$

Using the external definition of the gradient $\mathrm{d}f = \langle g(x), \mathrm{d}x\rangle$[1], we conclude:

$$\boxed{g(x) \triangleq \nabla f(Ax) = A^T \nabla_u f(Ax)\}}$$

2 Hessian of a composed vector function

Given $H_f(x)$, the Hessian of $f(x)$, and using the previous result, compute the Hessian of $f(A \cdot x)$.

[1] The meaning of "external definition" is that the gradient is not directly defined. Instead, we first posit that the **differential** of f can be expressed in a certain form, i.e. $\langle g(x), \mathrm{d}x\rangle$. If we find such a form, the gradient is defined to be $g(x)$.

Solution

A^T is a constant matrix, so we can write:

$$\mathrm{d}g = A^T \cdot \mathrm{d}\nabla_u f(Ax) = A^T \cdot \underbrace{\nabla_u^2 f(u)}_{H_{f(u)}=\text{Hessian of } f} \cdot \mathrm{d}u$$

Substitute u for Ax and get:

$$\mathrm{d}g = A^T \cdot \nabla_u^2 f(Ax) \cdot A \cdot \mathrm{d}x$$

And according to the external definition of the Hessian, we conclude:

$$\boxed{H(x) = A^T \cdot H_f(A \cdot x) \cdot A}$$

3 Gradient of a vector function composed with a scalar function

Compute the gradient of $f(x) = \varphi\left(h(x)\right)$, when: $\begin{cases} x \in \Re^n \\ h : \Re^n \to \Re \\ \varphi : \Re \to \Re \end{cases}$. All derivatives of $h(x)$ and of $\varphi(x)$ are given.

Solution

$\varphi(x)$ is a scalar function, therefore we can simply write (using the notation $u \triangleq h(x)$):

$$\mathrm{d}f = \mathrm{d}\left[\varphi\left(h(x)\right)\right] = \mathrm{d}\varphi(u) = \varphi'(u) \cdot \mathrm{d}u$$

After substitution we get:

$$\mathrm{d}f = \varphi'\left(h(x)\right) \cdot \mathrm{d}h(x) = \underbrace{\varphi'\left(h(x)\right)}_{\text{scalar}} \cdot \nabla h(x)^T \cdot \mathrm{d}x = \langle \varphi'\left(h(x)\right) \cdot \nabla h(x), \mathrm{d}x \rangle$$

As previously shown, using the external definition of the gradient, we conclude:

$$\boxed{\nabla f(x) = \varphi'\left(h(x)\right) \cdot \nabla h(x)}$$

.

4 Hessian of previous function

Compute the Hessian of the previous function.

Solution

We denote as $g(x)$ the gradient of $f(x)$, i.e. $g(x) = \varphi'(h(x)) \cdot \nabla h(x)$.

We seek to compute $\mathrm{d}g$. In order to do so, we will take each of the two members of the last expression, suppose it is constant, and apply the differential operator, and sum the two results (as in the well-known multiplication rule for derivatives):

$$\mathrm{d}g = \nabla h(x) \cdot \mathrm{d}\left[\varphi'(h(x))\right] + \varphi'(h(x)) \cdot \mathrm{d}\nabla h(x)$$

Remember $\varphi'(h(x))$ is a scalar. This means the order of the products in the last equation is not important.

We get:

$$\begin{aligned} \mathrm{d}g &= \nabla h(x) \cdot \varphi''(h(x)) \cdot [\nabla h(x)]^T \cdot \mathrm{d}x + \varphi'(h(x)) \cdot \underbrace{H_h(x)}_{\text{Hessian of h(x)}} \mathrm{d}x \\ &= \left[\varphi''(h(x)) \cdot \nabla h(x) \cdot \nabla h(x)^T + \varphi'(h(x)) \cdot H_h(x)\right] \cdot \mathrm{d}x \end{aligned}$$

We conclude:

$$\boxed{H_f = \varphi''(h(x)) \cdot \nabla h(x) \cdot \nabla h(x)^T + \varphi'(h(x)) \cdot H_h(x)}$$

5 Gradient of Neural Network function

The output of a neural network is $f(x, W_1, W_2) = \psi\left(v^T \cdot \phi_2(W_2 \cdot \phi_1(W_1 \cdot x)) - y\right)$, where $\psi(\cdot)$is a scalar function and $\phi_1, \phi_2 : \mathbb{R}^n \to \mathbb{R}^n$ are component-wise functions (refer to lecture for further details).

Compute the gradient of f with respect to W_1 and W_2.

Solution

We denote $s \triangleq W_1 \cdot x$, $t \triangleq W_2 \cdot \phi_1(s)$, $u \triangleq v^T \cdot \phi_2(t)$.

To compute the gradient with respect to W_1, we first suppose W_2 is constant. Note that:

$$du = v^T \cdot \phi_2' \cdot dt$$

$$dt = W_2 \cdot \phi_1' \cdot ds$$

We defined ϕ_1' as a diagonal matrix, which contains the components of $\nabla\phi_1$[2], i.e.:

[2] Actually, ϕ_1' is the Jacobian matrix of the vector function ϕ_1. The Jacobian matrix is a generalization of the gradient for functions from $\mathbb{R}^n$ to $\mathbb{R}^n$, when the gradient is defined only for functions from $\mathbb{R}^n$ to $\mathbb{R}$. In this specific case, ϕ_1 is a component-wise function, i.e. every scalar component of the output depends only on the relevant scalar component of the output. Only in this particular case the Jacobian matrix is diagonal and thus symmetric.

Informally, ϕ_1 is a set of uncorrelated scalar functions ($\mathbb{R} \to \mathbb{R}$) which are "combined" in a common vector for the sake of writing all the scalar equations as a single matrix equation.

$$\phi_1' \triangleq \begin{pmatrix} \frac{\partial \phi_1}{\partial s^{(1)}} & & & 0 \\ & \frac{\partial \phi_1}{\partial s^{(2)}} & & \\ & & \ddots & \\ 0 & & & \frac{\partial \phi_1}{\partial s^{(n)}} \end{pmatrix}$$

ϕ_2' is defined similarly.
Furthermore, recall that:

$$ds = dW_1 \cdot x$$

With this noted, we now remark:

$$df = \psi' \cdot du$$

Since both ψ' and du are scalars, this can be equivalently written as:

$$df = Tr(\psi' \cdot du)$$

We can now substitute the previous results:

$$df = Tr\left(\psi' \cdot v^T \cdot \phi_2' \cdot W_2 \cdot \phi_1' \cdot dW_1 \cdot x\right)$$

The trace is invariant under cyclic permutations, therefore:

$$\begin{aligned} df &= Tr\left(x \cdot \psi' \cdot v^T \cdot \phi_2' \cdot W_2 \cdot \phi_1' \cdot dW_1\right) \\ &= \langle \left(x \cdot \psi' \cdot v^T \cdot \phi_2' \cdot W_2 \cdot \phi_1'\right)^T, dW_1 \rangle \\ &= \langle \phi_1' \cdot W_2^T \cdot \phi_2' \cdot v \cdot \psi' \cdot x^T, dW_1 \rangle \end{aligned}$$

The last result comes from the fact that ϕ_1', ϕ_2' are diagonal and thus symmetric, and from the order-inverting property of the transpose operator.

We eventually conclude (from the external definition of the gradient) that the gradient of the function is:

$$\boxed{\nabla_{W_1} f = \phi_1' \cdot W_2^T \cdot v \cdot \psi' \cdot x^T}$$

This result is used in the backpropagation algorithm, whose purpose is to update the weights of a neural network.

We now seek to find the gradient with respect to W_2 .

W_1 is now considered to be constant. du and ds remain identical, but dt becomes:

$$dt = dW_2 \cdot \phi_1(W_1 x)$$

Therefore, in a similar way to the previous proof:

$$
\begin{aligned}
df &= Tr\left(\psi' \cdot v^T \cdot \phi_2' \cdot dW_2 \cdot \phi_1(W_1 \cdot x)\right) \\
&= Tr\left(\underbrace{\psi' \cdot \phi_1(W_1 \cdot x) \cdot v^T \cdot \phi_2'}_{\left(\nabla_{W_2} f\right)^T} \cdot dW_2\right)
\end{aligned}
$$

And we can conclude:

$$
\boxed{\nabla_{W_2} f = \psi' \cdot \phi_2' \cdot v \cdot \phi_1\left(W_1 \cdot x\right)}
$$

Note that $\phi_1\left(W_1 \cdot x\right)$is the input to the second layer.

Part II
Convex sets and functions

1 Convexity of norm function

A general norm function $\|\cdot\|$ fulfills the following conditions:

$$
\begin{array}{ll}
(1) & \|x\| \geq 0 \\
(2) & \|\alpha \cdot x\| = |\alpha| \cdot \|x\| \\
(3) & \|x\| = 0 \Leftrightarrow x = 0 \\
(4) & \|x + y\| \leq \|x\| + \|y\|
\end{array}
$$

Show that such a function is convex.

Solution

We first denote $f(x) \triangleq \|x\|$. Let $\lambda_1, ..., \lambda_n \geq 0$, such that $\sum \lambda_i = 1$ and let $x_1, ..., x_n$ be *any* vectors.

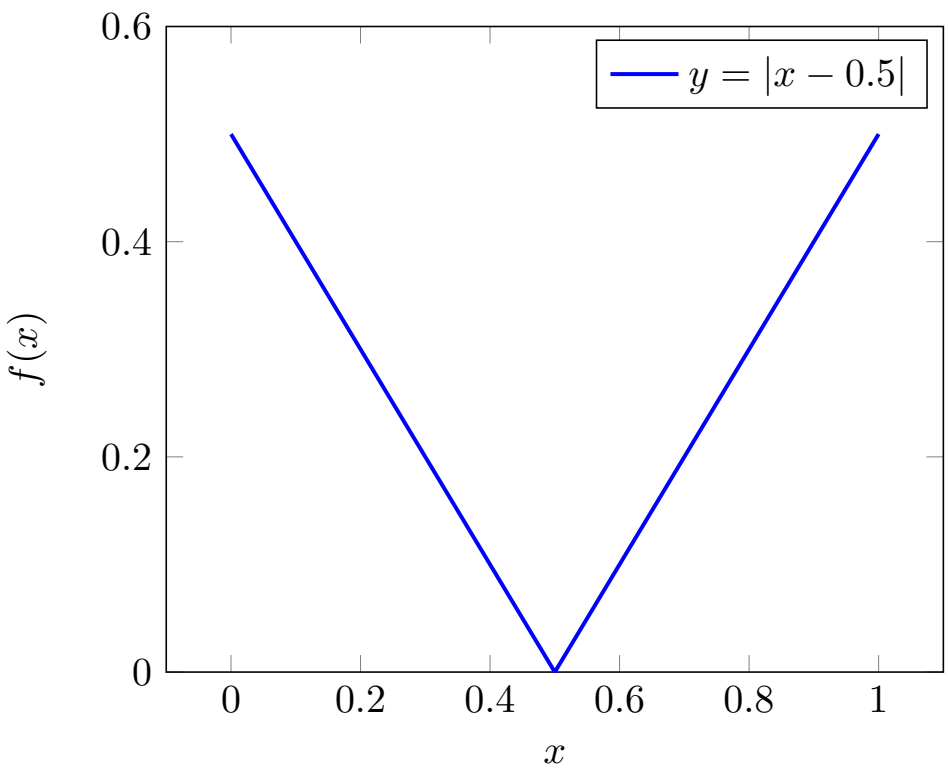
$$
f\left(\sum_i \alpha_i x_i\right) = \left\|\sum_i \lambda_i x_i\right\| \underset{(4)}{\leq} \sum_i \|\lambda_i x_i\| \underset{(2)}{=} \sum_i |\lambda_i| \cdot \|x_i\| \underset{\lambda_i \geq 0}{=} \sum_i \lambda_i f(x_i)
$$

Jensen's inequality holds true for any input, therefore the function is convex.

Part III
Iterative optimization algorithms

1 Golden section search

Run three iterations of the golden section algorithm on the function $f(x) = |x - \frac{1}{2}|^2$ in the interval $[0, 1]$.



Solution

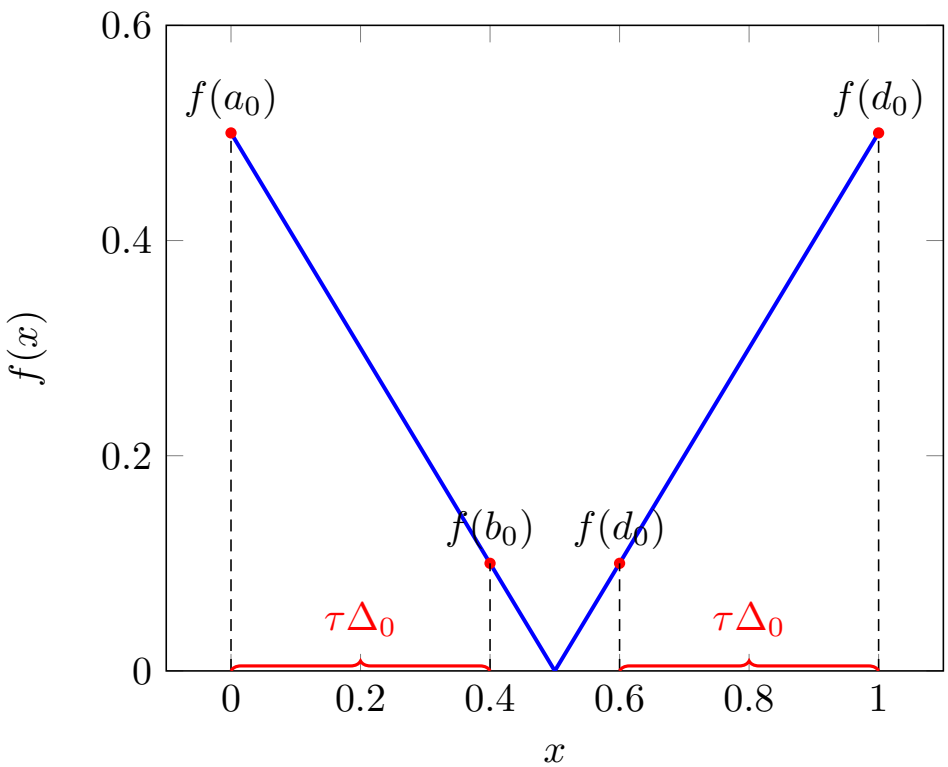
As usual, we denote $\tau \triangleq \frac{3-\sqrt{5}}{2} \approx 0.4$. Δ_n is the length of the considered interval $[a_n, d_n]$ at iteration n.

We recall succintly the iterative algorithm in pseudocode:

```
Choose base interval [a_0, d_0]
repeat
    Choose b_n right of a_n by τΔ_n
    Choose c_n left of d_n by τΔ_n
    if f(b_n) ≤ f(c_n) then
```

```
        Choose next interval "left" [a_n, c_n]
    else
        Choose next interval "right" [b_n, d_n]
until stop condition
```

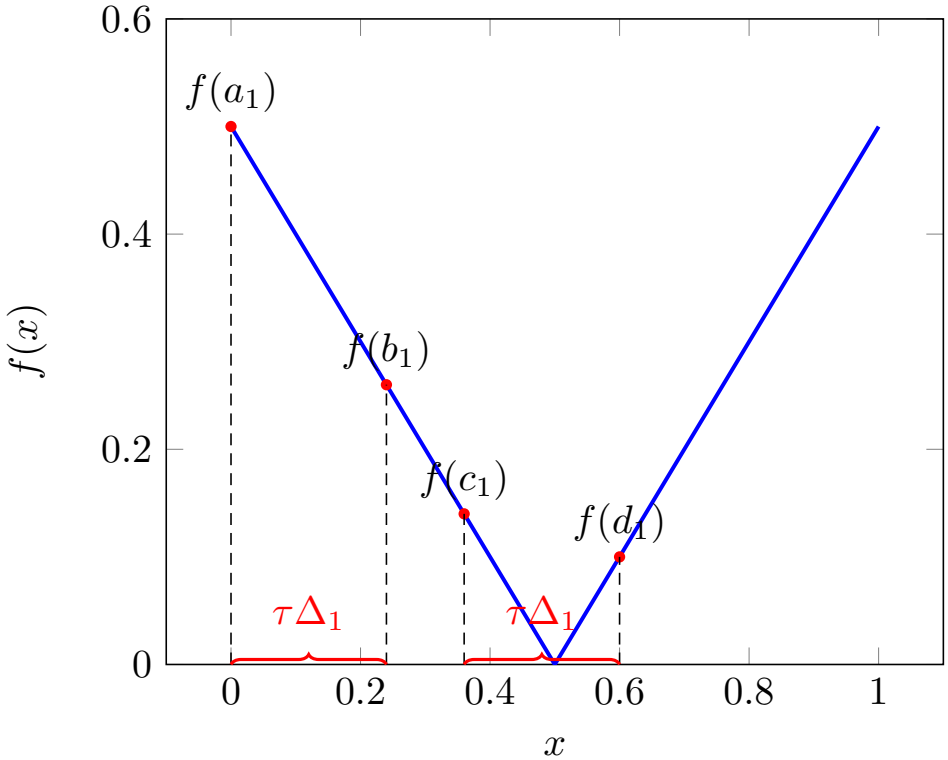
(Iteration 0) $a_0 = 0 \quad d_0 = 1 \quad b_0 = a_0 + \tau\Delta_0 \approx 0.4 \quad c_0 = a_0 + (1-\tau)\cdot\Delta_0 \approx 0.6$



We chose the interval $[a_0, c_0]$, i.e. $a_1 := a_0$, $d_1 := c_0$[3]. This leads to:

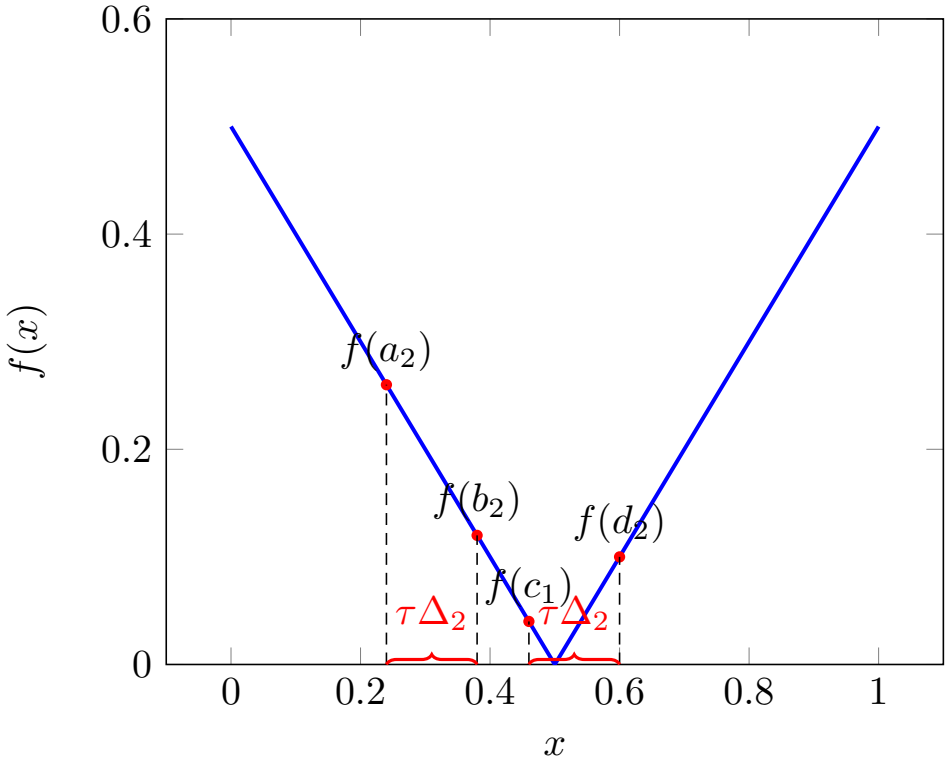
(Iteration 1) $a_1 = 0 \quad d_1 \approx 0.6 \quad b_1 = a_1 + \tau\Delta_1 \approx 0.24 \quad c_1 = a_1 + (1-\tau)\cdot\Delta_1 \approx 0.36$

[3]Since $f(b_0) = f(c_0)$, the choice of "going right" or "going left" is arbitrary, since in any case the minimum will be in the next interval. He we chose the left interval.



Since $f(b_1) < f(c_1)$, we will search next in $[b_1, d_1]$.As such, we assign $a_2 \coloneqq b_1$, $d_2 \coloneqq d_1$ ($\Delta_2 \approx 0.36$):

$$\text{(Iteration 2)} \quad a_2 = 0.24 \quad d_2 \approx 0.6 \quad b_2 = a_2 + \tau\Delta_2 \approx 0.38 \quad c_2 = a_2 + (1-\tau)\cdot\Delta_2 \approx 0.46$$



The next interval would be $[b_2, d_2]$.

2 Steepest descent

The Rosenbrock function is ($x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$):

$$f(x) = (1 - x_1)^2 + 100 \cdot (x_2 - x_1^2)^2$$

What will be the convergence rate of gradient descent around the solution point: $s = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$?

Solution

$$f(x) = 1 - 2x_1 + x_1^2 + 100\left(x_2^2 - 2x_2x_1^2 + x_1^4\right)$$

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} = \begin{pmatrix} -2 + 2x_1 + 100 \cdot (-2x_2 \cdot 2x_1 + 4x_1^3) \\ 100 \cdot (2x_2 - 2x_1^2) \end{pmatrix}$$

$$
\begin{aligned}
H(x) &= \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} \\
&= \begin{pmatrix} 2 + 100 \cdot (-4x_2 + 4 \cdot 3 \cdot x_1^2) & -400x_1 \\ -400x_1 & 200 \end{pmatrix} \\
&= \begin{pmatrix} 2 - 400x_2 + 1200x_1^2 - 400x_1 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}
\end{aligned}
$$

At the solution point s:

$$
H(s) = \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix}
$$

We need to find the two eigenvalues λ_1, λ_2 of the Hessian matrix. We will use two well-known relations to find them:

$$
\begin{array}{ll}
(*) & Tr(H) = \sum_i \lambda_i = \lambda_1 + \lambda_2 \\
(**) & \det(H) = \prod_i \lambda_i = \lambda_1 \lambda_2
\end{array}
$$

We get the system:

$$
\begin{cases} \lambda_1 + \lambda_2 & = 802 + 200 \\ \lambda_1 \lambda_2 & = 802 \cdot 200 - 400^2 \end{cases} \iff \begin{cases} \lambda_1 + \lambda_2 & = 1002 \\ \lambda_1 \lambda_2 & = 400 \end{cases}
$$

Using substitution and the quadratic formula:

$$
\begin{cases} \lambda_1 & = 501 + \sqrt{250601} \\ \lambda_2 & = 501 - \sqrt{250601} \end{cases}
$$

As noted in the lecture, the convergence rate is $c = 1 - \frac{\lambda_{min}}{\lambda_{max}}$. Thus:

$$
\boxed{c = 1 - \frac{501 - \sqrt{250601}}{501 + \sqrt{250601}} \approx 0.9996}
$$

3 Continuation of previous exercise - Convergence rate

In the previous exercise we found that the convergence rate of the steepest gradient method applied on the Rosenbrock function around the solution is:

$$
c = 1 - \frac{501 - \sqrt{250601}}{501 + \sqrt{250601}} \approx 0.9996
$$

How many iterations are needed to improve the accuracy by 10? I.e., if we found an iterative solution at the step k_0 near the "real" solution f^*, which satisfies:

$$f_{k_0} - f^* = a$$

Then what is the number k_1 which satisfies:

$$f_{k_1} - f^* = 0.1a$$

What will be the solution if instead of the steepest gradient algorithm we use the conjugate gradient algorithm?

Solution

In the general case, if we denote the convergence rate as c, we obtain:

$$f_{k+1} - f^* \leq c \cdot (f_k - f^*)$$

$$f_{k+2} - f^* \leq c^2 \cdot (f_k - f^*)$$

After i iterations:

$$f_{k+i} - f^* \leq c^i \underbrace{(f_k - f^*)}_{a}$$

The requirement is $c^i \stackrel{!}{=} 0.1$.
We apply the logarithm function on both sides of the equation:

$$i \cdot \log_{10}(c) = -1$$

$$i = -\frac{1}{\log_{10}(c)} = -\frac{\ln(10)}{\ln(c)} = \frac{\ln(0.1)}{\ln(c)}$$

And we get the general formula:

$$\boxed{i = \log_c 0.1}$$

In the case of the steepest descent method:

$$i = \log_{0.9996} = 5755$$

In the case of the conjugate gradient method the convergence rate is:

$$c = 1 - \sqrt{\frac{\lambda_{min}}{\lambda_{max}}} \approx 0.98$$

So that the final answer is:

$$\log_{0.98} 0.1 = 114$$

This shows the conjugate gradient method is much more efficient.

4 Newton and Gauss-Newton methods

Our goal is to efficiently solve the set of linear equations:

$$A \cdot x = b$$

where A is a matrix of size n

Directly solving the set of equations (i.e. inversing the matrix A) is a costly operation[4].

Given the modified Cholevski Decomposition of the matrix A:

$$A = L \cdot D \cdot L^T$$

(where D is a diagonal matrix and L is a lower unit triangular matrix), is it possible to solve more efficiently the set of equations?

Note:

- Inversing a diagonal matrix is done by inversing each one of the members of the diagonal. Thus its complexity is only $O(n)$.
- Inversing a triangular matrix is also considered as simple[5] .

Solution

The system is:

$$L \cdot D \cdot L^T \cdot x = b$$

We denote $y \triangleq DL^T x$.

We get

$$L \cdot y = b$$

And we easily find y by inversing L.

We now denote $z \triangleq L^T x$, therefore:

$$y = D \cdot z$$

We can easily inverse a diagonal matrix, thus:

$$z = D^{-1} \cdot y$$

We finally need to solve:

$$L^T \cdot x = z$$

[4] $O\left(n^3\right)$ in a straighforward way, or $O\left(n^{2.373}\right)$ using optimized algorithms. See Wikipedia article on Computational complexity of mathematical operations.

[5] At most $O\left(n^2\right)$. See M. Malek, Inverse of Lower Triangular Matrices. Moreover, the fact that all the members are 0 or 1 can be exploited to greatly reduce the number of needed computations.

Which is similarly easy to solve, since L^T is a upper unitriangular matrix and its inverse is the transpose of L^{-1}, which we have already computed.

Summary:

1. Compute $y = L^{-1} \cdot b$.
2. Compute $z = D^{-1} \cdot y$.
3. Compute $x = L^{-T} \cdot z$, where $L^{-T} = \left(L^T\right)^{-1} = \left(L^{-1}\right)^T$.

Part IV
Mathematical definitions

1 Scalar product

Show that the binary operator $(\cdot\,,\cdot)_Q$, which is defined as follows:

$$(a, b)_Q \triangleq x^T Q y$$

when $Q \succ 0$, is a scalar product (i.e. satisfies all the properties listed in the lecture).

Solution

For real numbers:

$$\begin{array}{ll}
(1) & (x, y) = x^T Q y = \left(y^T Q x\right)^T \overset{\text{scalar}}{=} \left(y^T Q x\right) = (y, x) \\
(2) & (x, y + z) = x^T Q \cdot (y + z) = x^T Q y + x^T Q y = (x, y) + (x + z) \\
(3) & (\alpha \cdot x, y) = (\alpha \cdot x)^T Q y = \alpha \cdot x^T Q y = \alpha \cdot (x, y) \\
(4, 5) & (x, x) = x^T Q x > 0 \ \forall x
\end{array}$$

In (1) we used the fact that Q is positive-definite matrix, and thus symmetric, i.e. $Q^T = Q$.

The last line derives directly from the positive definitiveness of Q.

For complex numbers[6], we can demonstrate similar properties.

[6] See the Wikipedia article on positive-definite matrix for an extension of the definition of semidefinite matrices to complex matrices and vectors. The matrix is then Hermitian, which is a generalization of a real symmetric matrix.

However, note that the exact generalization of the inner product definition to complex numbers is a matter of debate/application - see Wikipedia articles on inner product space and sesquilinear form.

$$
\begin{array}{ll}
(1) & (x,y) = x^*Qy = x^*Q^*y = (y^*Qx)^* = (y,x)^* \\
(2) & (x,y+z) = x^*Q \cdot (y+z) = x^*Qy + x^*Qy = (x,y) + (x+z) \\
(3) & (\alpha \cdot x, y) = (\alpha \cdot x)^*Qy = \alpha^* \cdot x^*Qy = \alpha^* \cdot (x,y) \\
(4,5) & (x,x) = x^*Qx > 0 \ \forall x
\end{array}
$$

2 Graham-Schmidt method

Real-valued functions are a vector space. The scalar product between two vectors in this space (i.e. functions) is defined as follows:

$$\langle f, g \rangle = \int_A f(x)g(x)dx$$

Given the three following vectors, build an orthogonal set using the Graham-Schmidt method:

$$
\begin{array}{rcl}
f_0 & = & 1 \\
f_1 & = & x \\
f_2 & = & x^2
\end{array}
$$

Solution

$$y_0 = f_0 \boxed{= 1}$$

$$
\begin{aligned}
y_1 &= f_1 - \frac{\langle f_1, y_0 \rangle}{\langle y_0, y_0 \rangle} y_0 \\
&= x - \frac{\int_0^1 x \cdot 1 \cdot dx}{\int_0^1 1 \cdot 1 \cdot dx} \cdot 1 \boxed{= x - \frac{1}{2}}
\end{aligned}
$$

$$
\begin{aligned}
y_2 &= f_2 - \frac{\langle f_2, y_1 \rangle}{\langle y_1, y_1 \rangle} y_1 - \frac{\langle f_2, y_0 \rangle}{\langle y_0, y_0 \rangle} y_0 \\
&= x^2 - \frac{\int_0^1 x^2 \cdot \left(x - \frac{1}{2}\right) \cdot dx}{\int_0^1 \left(x - \frac{1}{2}\right)^2 \cdot dx} \cdot \left(x - \frac{1}{2}\right) - \frac{\int_0^1 x^2 \cdot 1 \cdot dx}{\int_0^1 1 \cdot 1 \cdot dx} \cdot 1 \\
&= x^2 - \frac{1/4 - 1/2 \cdot 1/3}{1/3 - 1/2 + 1/4} \left(x - \frac{1}{2}\right) - \frac{1}{3} \\
&= \boxed{x^2 - x + \frac{1}{6}}
\end{aligned}
$$

Part V
Advanced multidimensional iterative algorithms

1 Find minimum of quadratic form using Complex Conjugate Method

Substitute

$$x = \sum_{i=1}^{n} \alpha_i d_i$$

in

$$f(x) = \frac{1}{2} x^T Q x + b^T x$$

to get the final expression:

$$f(x) = \sum_{i=1}^{n} \left(\frac{1}{2} \alpha_i^2 \left\| d_i \right\|_Q^2 + \alpha_i b^T d_i \right)$$

When $\|\cdot\|_Q$ has been defined on page 14.

Solution

$$\frac{1}{2} \left(\sum_{i=1}^{n} \alpha_i d_i \right) Q \left(\sum_{i=1}^{n} \alpha_i d_i \right) + b^T \sum_{i=1}^{n} \alpha_i d_i = \frac{1}{2} \sum_i \alpha_i d_i^T Q \left(\sum_i \alpha_i d_i \right) + \sum_i \alpha_i b^T d_i$$

$$= \sum_i \frac{1}{2} \alpha_i^2 \underbrace{d_i^T Q d_i}_{=\|d_i\|^2} + \sum \alpha_i b^T d_i = \sum_i \frac{1}{2} \alpha_i^2 \left\| d_i \right\|_Q^2 + \alpha_i b^T d_i$$

2 Preconditioning

The following matrix is given:

$$Q = \begin{pmatrix} 100 & 1 \\ 1 & 15625 \end{pmatrix}$$

Since the diagonal values of Q are "far greater" than the rest of its entries in each row, Q is called "diagonal-dominant". We wish to transform the problem $Qx = b$ into a form that will converge quickier. This is called "preconditioning". Since the convergence rate depends on thecondition number of the matrix,

our goal is to reduce this property of the system. Follows an exemple of such preconditioning.

We define the matrix:

$$S = \left(\sqrt{diag(Q)}\right)^{-1} = \begin{pmatrix} \frac{1}{\sqrt{Q_{11}}} & & & \\ & \frac{1}{\sqrt{Q_{22}}} & & \\ & & \ddots & \\ & & & \frac{1}{\sqrt{Q_{nn}}} \end{pmatrix}$$

What are the respective condition numbers of Q and of $\tilde{Q} \triangleq S^T Q S$?

Solution

For PSD and thus symmetric matrices (as a special case of normal matrices) the condition number is defined as[7]:

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$$

For Q :

$$\lambda_1 \approx 100$$

$$\lambda_2 \approx 15625$$

$$\Rightarrow \kappa\left(Q\right) \approx \frac{15625}{100} = 156.25$$

We now explicitly compute S:

$$S = \begin{pmatrix} \frac{1}{\sqrt{100}} & 0 \\ 0 & \frac{1}{\sqrt{15625}} \end{pmatrix}$$

And then we compute $\tilde{Q}$:

$$\tilde{Q} = \begin{pmatrix} \frac{1}{\sqrt{100}} & 0 \\ 0 & \frac{1}{\sqrt{15625}} \end{pmatrix} \begin{pmatrix} 100 & 1 \\ 1 & 15625 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{100}} & 0 \\ 0 & \frac{1}{\sqrt{15625}} \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{1250} \\ \frac{1}{1250} & 1 \end{pmatrix}$$

It is nearly diagonal, so we expect its eigenvalues to be approximatively $\{1, 1\}$. Indeed, if we denote $a \triangleq 100$, $b \triangleq 15625$, we find (after a simple computation of the eigenvalues, see Class 3):

$$\lambda_{1,2} = 1 \pm \frac{1}{\sqrt{a \cdot b}} = \{1.0008, 0.9992\}$$

[7] Strictly speaking the formula is absolute-valued, but since the matrix is PSD its eigenvalues are anyway positive.

We finally conclude:

$$\kappa\left(\tilde{Q}\right) = \frac{1.0008}{0.9992} \approx 1.002 \ll \kappa\left(Q\right)$$

The preconditioning has achieved its desired effect.

3 Alternating Direction Method of Multipliers (ADMM) with scaled dual variables

The ADMM method with scaled dual variables (see here, slide 17) is defined for a separable problem as follows:

$$\begin{array}{ll} \min & f(x) + g(z) \\ \text{s.t.} & Ax + Bz = c \end{array}$$

In this exercise we will apply this method in order to solve the generic constrained convex optimization problem:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in C \end{array}$$

When C is a convex set.

To do so, we define $g(\cdot)$ to be the indicator function of C, i.e the function that indicates membership of an element in C, in the following way[8]:

$$g(u) = \begin{cases} 0 & u \in C \\ \infty & u \notin C \end{cases}$$

We define a new variable z (which is called a "virtual variable"), and we define the following problem, which is equivalent to the original problem:

$$\begin{array}{ll} \min & f(x) + g(z) \\ \text{s.t.} & x - z = 0 \end{array}$$

Compute the augmented Lagrangian of this problem, and deduce the update formula.

Solution

In the general case, the augmented Lagragian with scaled dual variables is:

$$\mathcal{L}_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2} \left\| Ax + Bz - c + u \right\|_2^2$$

In our case $A = -B = I$ and $c = 0$, thus:

[8] In other words, the function "forces" the variable to be inside the set - otherwise the function value is infinity and the minimizing problem is unsolvable.

$$\mathcal{L}_\rho = f(x) + g(z) + \frac{\rho}{2}\|x - z + u\|_2^2$$

The general update formula is then (see slides linked in exercise):

$$\begin{aligned}
x^{(k+1)} &= \underset{x}{\operatorname{argmin}} \left\{ f(x) + \frac{\rho}{2}\left\|x - z^{(k)} + u^{(k)}\right\|_2^2 \right\} \\
z^{(k+1)} &= \underset{z}{\operatorname{argmin}} \left\{ g(z) + \frac{\rho}{2}\|z - a\|_2^2 \right\} \\
u^{(k+1)} &= u^{(k)} + x^{(k+1)} - z^{(k+1)}
\end{aligned}$$

When we denoted $a \triangleq x^{(k+1)} + u^{(k)}$.

In other words (since $g(z)$ is an indicator function, whose only purpose is to force z to be in C):

$$\begin{aligned}
z^{(k+1)} = \quad & \underset{z}{\operatorname{argmin}} \quad \|z - a\|_2^2 \\
& \text{s.t.} \qquad z \in C
\end{aligned}$$

That is, $z^{(k+1)}$ is the closest point to a in the set C. Evidently, the solution is the projection of a onto C. Therefore:

$$z^{(k+1)} = \operatorname{Proj}_C\left(x^{(k+1)} + u^{(k)}\right)$$

And the last iterative step for u is, as said above:

$$u^{(k+1)} = u^{(k)} + x^{(k+1)} - z^{(k+1)}$$

4 Lasso Problem using regular ADMM

Use a method similar to the previous exercise to solve the Lasso problem using the *regular* ADMM algorithm :

$$\text{minimize} \quad \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

Solution

We again define a virtual variable z and define the new equivalent problem:

$$\begin{aligned}
\min \quad & \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda\|z\|_1 \\
\text{s.t.} \quad & x - z = 0
\end{aligned}$$

The augmented Lagrangian is then:

$$\begin{aligned}
\mathcal{L}_\rho(x, z, y) &= f(x) + g(z) + y^T(x - z) + \frac{\rho}{2}\|x - z\|_2^2 \\
&= \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|z\|_1 + y^T(x - z) + \frac{\rho}{2}\|x - z\|_2^2
\end{aligned}$$

We then compute the update formula (after deleting the elements depending on z only, and which don't affect the argmin function):

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \frac{1}{2} \|Ax - b\|_2^2 + y^T x + \frac{\rho}{2} \left\| x - z^{(k)} \right\|_2^2 \right\}$$

To find the minimum of this unconstrained optimization problem, we compute the gradient of the augmented Lagrangian and equate it to zero:

$$\nabla_x \mathcal{L}(x, z, y) = A^T (Ax - b) + y + \rho(x - z^{(k)}) \stackrel{!}{=} 0$$

$$\Rightarrow \left(A^T A - \rho I\right) x = \rho z^{(k)} - y + A^T b$$

$$\Rightarrow \boxed{x^{(k+1)} = \left(A^T A - \rho I\right)^{-1} \cdot \left(\rho z^{(k)} - y + A^T b\right)}$$

We now compute the update formula for z:

$$z^{(k+1)} = \underset{z}{\operatorname{argmin}} \left\{ \lambda \|z\|_1 - \left(y^{(k)}\right)^T z + \frac{\rho}{2} \left\| x^{(k+1)} - z \right\|_2^2 \right\} = \underset{z}{\operatorname{argmin}} \left\{ \sum_i |z_i| \cdot \lambda - y_i^{(k)} z_i + \frac{\rho}{2} (x_i^{(k+1)} - z_i)^2 \right\}$$

Since the elements of the sum are independent , we can solve separately for each component of the vector z:

$$z_i^{(k+1)} = \underset{z_i}{\operatorname{argmin}} |z_i| \cdot \lambda - y_i^{(k)} z_i + \frac{\rho}{2} (x_i^{(k+1)} - z_i)^2$$

This problem can be solved using the soft thresholding method, whose proof we will quickly sketch (inspired by Simon Lucey's proof):

1. Suppose the solution is strictly positive, i.e. $z_i > 0$. We derive the function according to z_i , we equate to zero and finally obtain the optimum point $z_i^* = x_i + \frac{y_i}{\rho} - \frac{\lambda}{\rho}$.
2. Suppose $z_i < 0$. We get the optimum point $z_i^* = x_i + \frac{y_i}{\rho} + \frac{\lambda}{\rho}$.
3. Suppose $z_i = 0$. The only solution is of course $z_i^* = 0$.

We now check under which condition every solution fits the corresponding assumption.

1. $z_i^* = x_i + \frac{y_i}{\rho} - \frac{\lambda}{\rho} > 0 \Rightarrow x_i + \frac{y_i}{\rho} > \frac{\lambda}{\rho}$.
2. $z_i^* = x_i + \frac{y_i}{\rho} + \frac{\lambda}{\rho} < 0 \Rightarrow x_i + \frac{y_i}{\rho} < -\frac{\lambda}{\rho}$.
3. Following the two last results, this solution is true if $-\frac{\lambda}{\rho} \leq x_i + \frac{y_i}{\rho} \leq \frac{\lambda}{\rho} \Leftrightarrow \left| x_i + \frac{y_i}{\rho} \right| \leq \frac{\lambda}{\rho}$.

We can summarize the result as:

$$z_i^{(k+1)} = \begin{cases} x_i + \frac{y_i}{\rho} - \frac{\lambda}{\rho} & x_i + \frac{y_i}{\rho} > \frac{\lambda}{\rho} \\ x_i + \frac{y_i}{\rho} + \frac{\lambda}{\rho} & x_i + \frac{y_i}{\rho} < -\frac{\lambda}{\rho} \\ 0 & \left| x_i + \frac{y_i}{\rho} \right| \leq \frac{\lambda}{\rho} \end{cases}$$

This function is called the "soft thresholding operator".