Minimizing an arbitrary function:

Algorithm 1 Damped Newton's Method

Require: x_0

1: **for** $k \geq 0$ **do**
2: Compute direction p_k, where $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$
3: Find α_k from Line Search (Given in pg 2)
4: $x_k \leftarrow x_k + \alpha_k p_k$
5: STOP when STOPPING CRITERION satisfied
6: **end for**

Condition: $\nabla^2 f(x) > 0$ (i.e. hessian is +ve def.)

Algorithm ~~1~~ 2 Gradient Descent

Require: x_0

1: **for** $k \geq 0$ **do**
2: Compute direction p_k, where $p_k = -\nabla f(x_k)$
3: Find α_k from Line Search (Given in pg 2)
4: $x_k \leftarrow x_k + \alpha_k p_k$
5: STOP when STOPPING CRITERION satisfied
6: **end for**

The stop criterion could be:

- max no of iterations;
- proximity of α_{k+1} to x_k;
- how smaller the gradient at α_{k+1} gets.

Algorithm 3.1 (Backtracking Line Search).

Choose $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$; Set $\alpha \leftarrow \bar{\alpha}$;
repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
$\alpha \leftarrow \rho\alpha$;
end (repeat)
Terminate with $\alpha_k = \alpha$.

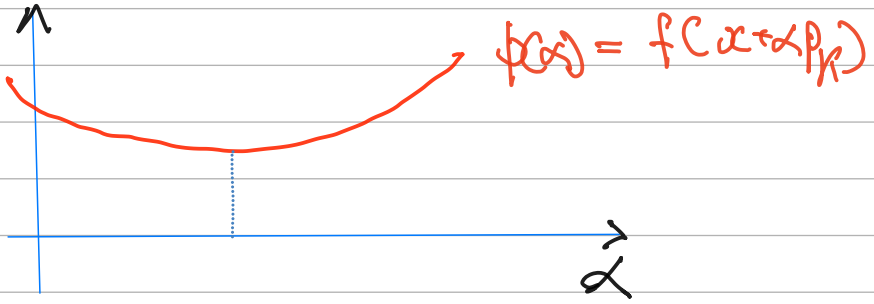
what is $\bar{\alpha}$?

Exact line Search.

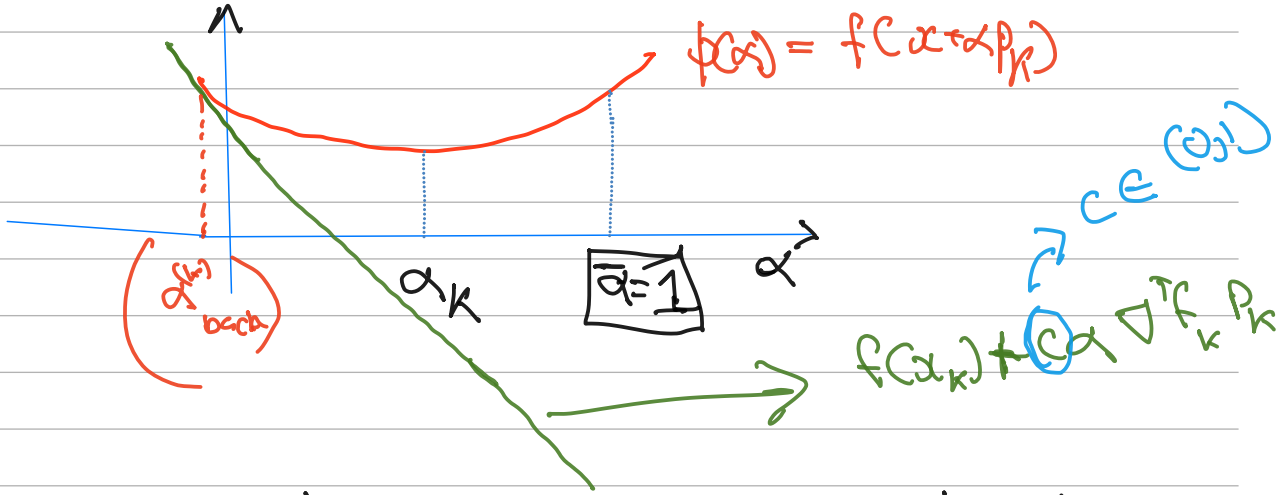
So we have; $\alpha_k = \underset{\alpha>0}{\operatorname{argmin}} f(x+\alpha p_k)$ → $\phi(\alpha)$

Basically what we do is exhaustively find $\phi(\alpha)$ and pick α_k as from the smallest $\phi(\alpha)$.

Supose our function $\phi(\alpha)$ looks like below—



Backtracking.



In backtracking we choose an $\bar{\alpha}$ (a guess) as initial value of α aka steplength, such that $\alpha > 0$. Let $\alpha = 1$ for above example.

Now we have a condition; $\phi(\alpha) = f(x+\alpha p_k) > f(x_k) + c\alpha \nabla f_k^T p_k$

As long as the above condition is valid we multiply α by β.

$$\alpha_{n+1} = \beta \alpha_n$$

β can change in each iteration like $\beta, \beta^2, \beta^3, \ldots$

where $\beta \in (0,1)$

β → contraction factor

Red function Green Line.

We do it till we violate the condition ie; LHS ≤ RHS

Then we assign $\alpha_k = \alpha_n$ (last iterated α that passed the condition not α_{n+1} which violated.

Side Note

Taylor series:

$$f(x) = f(x_0) + \sum_{n=1}^{n} \frac{(x-x_0)^n f^n(x)}{n!}$$

Newtons Raphson Method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Sequential Quadratic Programming :-

Lets see the KKT conditions of optimality. For a given function;

$$\min_x f(x) \quad ST \quad g(x) = 0, \quad h(x) \le 0$$

equality constraint
inequality constraint

The Lagrangian $\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$

KKT conditions :

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$$
$$g(x^*) = 0$$
$$h(x^*) \le 0$$
$$\mu_i \ge 0 \ \forall i$$
$$\mu_i \cdot h_i(x^*) = 0 \ \forall i$$

ie; if the constraint is active only then will it have non-zero multipliers.

Now lets consider another function with one equality constraint;

$$\min_x f(x) = 0 \quad ST \quad g(x) = 0$$

$L = f(x) + \lambda^T g(x)$ → $\lambda^T \nabla g(x)$ as per text

Its KKT is; $\nabla_x L(x) = \nabla f(x) + \nabla g(x)^T \lambda = 0$ — (1)

$g(x) = 0$ — (2)

For SQP :-

Differentiating ① $\nabla^2 f(x) + \lambda^T \nabla^2 g(x)$

② $\nabla g(x) = 0$

But the before notes and textbook writes this gradient as:

$$\nabla^2 f(x) + \nabla g(x)^T \cdot \lambda.$$

From slide the SQP is given by; → for descent

General Form

$$\begin{bmatrix} \text{Hess}(L) & \text{constraint grad.} \\ (\text{constraint gradient})^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = -\text{ve}(\text{gradient}(L)) \\ \text{constraint grad}$$

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k) & \nabla g(x_k) \\ \nabla g(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} p_k \\ p_\lambda \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k \\ -g(x_k) \end{pmatrix}$$

For our problem;

$$\begin{bmatrix} \nabla^2_{xx} L(x_k) & \nabla g(x_k)^T \\ \nabla g(x_k) & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k \\ -g(x_k) \end{bmatrix} \quad (M)$$

We have from Newton iterate step;

$$x_{k+1} = x_k + \alpha_k p_k$$

Clearly eqn.

lets wait on α_k for now

For our problem we can re write as;

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix}$$

$$\Rightarrow \nabla^2_{xx} L(x_k) \cdot p_k + \nabla g(x_k) \cdot p_\lambda = -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k$$

$$\nabla^2_{xx} L(x_k) \cdot p_k + \nabla g(x_k)^T \cdot \lambda_{k+1} - \nabla g(x_k)^T \cdot \lambda_k = -\nabla f(x_k) - \nabla g(x_k)^T \lambda_k$$

$$\Rightarrow \nabla^2_{xx} L(x_k) \cdot p_k + \nabla g(x_k)^T \cdot \lambda_{k+1} = -\nabla f(x_k)$$ PRIMAL UPDATE EQUATION

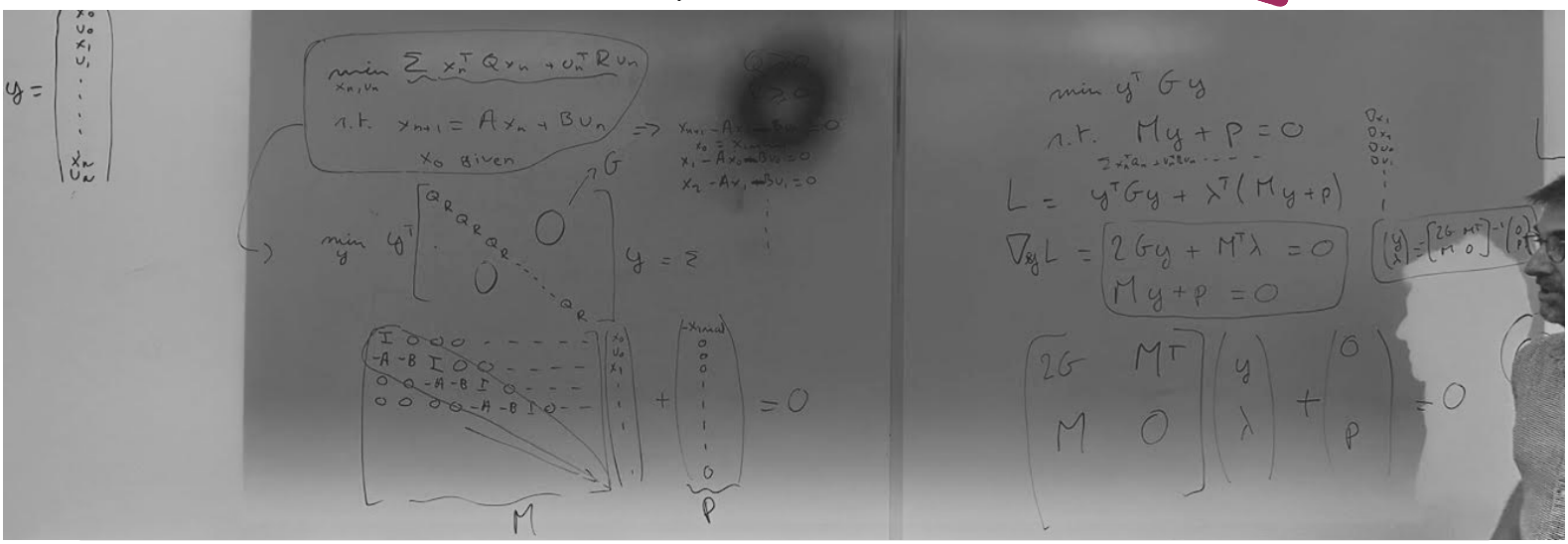
$$\rightarrow \nabla g(x_k) \cdot p_k = -g(x_k)$$ CONSTRAINT EQUATION

Hence we can rewrite the equations in a matrix form;

$$\begin{bmatrix} \nabla^2_{xx} L(x_k) & \nabla g(x_k)^T \\ \nabla g(x_k) & 0 \end{bmatrix} \cdot \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -g(x_k) \end{bmatrix} \rightarrow M$$

Note: we have already found in lecture 2 a similar form matrix. See figure

$$\begin{bmatrix} 2G & M^T \\ M & 0 \end{bmatrix} \begin{bmatrix} y \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ -p \end{bmatrix}$$



By that we can recompute m^1 as;

$$2\nabla^2_{xx}\mathcal{L}(x_k)p_k + \nabla g(x_k)^T \lambda_{k+1} = -\nabla f(x_k) \quad \rightarrow (1)$$

$$\nabla g(x_k)\cdot p_k = -g(x_k)$$

$\Rightarrow \int (1)$ wrt $p_k \Rightarrow p_k^T \nabla^2_{xx}\mathcal{L}(x_k)p_k +$

Step-by-Step Integration:

1. **Rearrange the Original Equation:**

$$2\,\nabla^2_{xx}\mathcal{L}\,p_k = -\nabla f(x_k) - \nabla g(x_k)^T \lambda_{k+1}$$

Here, for simplicity, let's denote $\nabla^2_{xx}\mathcal{L}(x_k, \lambda_{k+1})$ as H, and $\nabla f(x_k)$ as g. The equation becomes:

$$2Hp_k = -g - \nabla g(x_k)^T \lambda_{k+1}$$

2. **Isolate p_k:**

$$p_k = -\frac{1}{2}H^{-1}\left(g + \nabla g(x_k)^T \lambda_{k+1}\right)$$

However, for integration purposes, we focus on the relationship before solving for p_k.

3. **Integrate Both Sides with Respect to p_k:**

The left side is:

$$\int (2Hp_k)\, dp_k = Hp_k^2 + C_1$$

The right side involves integrating constants (with respect to p_k):

$$\int \left(-g - \nabla g(x_k)^T \lambda_{k+1}\right) dp_k = -gp_k - \nabla g(x_k)^T \lambda_{k+1} p_k + C_2$$

Combining both integrals:

$$Hp_k^2 + C_1 = -gp_k - \nabla g(x_k)^T \lambda_{k+1} p_k + C_2$$

Ignoring the constants of integration (since they can be absorbed or set to zero for optimization purposes), we get:

$$Hp_k^2 = -gp_k - \nabla g(x_k)^T \lambda_{k+1} p_k$$

However, to align with the desired form, we need to adjust the coefficients appropriately.

4. **Adjusting the Coefficients:**

   Notice that the quadratic term in p_k should have a coefficient of $\frac{1}{2}$. To achieve this, we divide both sides of the integrated equation by 2:

$$\frac{1}{2}Hp_k^2 = -\frac{1}{2}gp_k - \frac{1}{2}\nabla g(x_k)^T\lambda_{k+1}p_k$$

   Rearranging terms, the expression becomes:

$$\frac{1}{2}p_k^T\nabla_{xx}^2\mathcal{L}(x_k,\lambda_{k+1})p_k + p_k^T\nabla f(x_k) = \text{constant}$$

   Ignoring the constant (as it's not essential for the optimization step), we arrive at:

$$\frac{1}{2}p^T\nabla_{xx}^2\mathcal{L}(x_k)p + p^T\nabla f(x_k)$$

Conclusion:

By integrating the original linear equation with respect to p_k and appropriately adjusting the coefficients, we obtain the quadratic expression:

$$\frac{1}{2}p^T\nabla_{xx}^2\mathcal{L}(x_k)p + p^T\nabla f(x_k)$$

This quadratic form is often used in optimization, particularly in methods like Newton's method, where the goal is to find a search direction p that minimizes the quadratic approximation of the objective function.

ie; we get

$$\min_p \ \frac{1}{2}p^T\nabla_{xx}^2\mathcal{L}(x_k)p + p^T\nabla f(x_k)$$

$$\text{S.T. } \nabla g(x_k)^T p + g(x_k) = 0$$

Sequential Quadratic Programming

Given $\min_x f(x)$ S.T. $g(x) = 0$

① $\nabla(\min_x f(x)) = \nabla f(x)$.

② $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$

$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + \lambda^T \nabla g(x)$

Find. $\nabla^2_{xx} \mathcal{L}(x, \lambda)$

③ Make a matrix eqn of form

$$\begin{bmatrix} \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) & \nabla g(x_k)^T \\ \nabla g(x_k) & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -g(x_k) \end{bmatrix}$$

A $\quad$ x $\quad$ b

use `np.linalg.solve(A, b)` to get $x \Rightarrow \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix}$

From which obtain p_k.

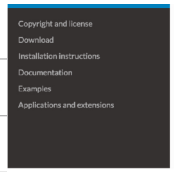
OR

OR

Step 1: Find a direction

$$\min_p \; p^T \nabla^2_{xx} \mathcal{L}(x_k) p + p^T \nabla f(x_k)$$
$$\text{subject to } \nabla g(x_k)^T p + g(x_k) = 0$$
$$\nabla h(x_k)^T p + h(x_k) \le 0$$

Use any QP solver!



CVXOPT

PYTHON SOFTWARE FOR CONVEX OPTIMIZATION

CVXOPT is a free software package for convex optimization based on the Python programming language. It can be used with the interactive Python interpreter, on the command line by executing Python scripts, or integrated in other software via Python extension modules. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Python's extensive standard library and on the strengths of Python as a high-level programming language.

Quadratic Programming

The function `qp` is an interface to `coneqp` for quadratic programs. It also provides the option of using the quadratic programming solver from MOSEK.

`cvxopt.solvers.qp(P, q[, G, h[, A, b[, solver[, initvals]]]])`

Solves the pair of primal and dual convex quadratic programs

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

④ Find step length α_k with line search

use merit function for line search

Create a function phi's:

```
def phi(x, f, g, μ)
    f_val = f       (cost)
    g_val = g       (constraints)
    l1norm = np.sum(np.abs(g_val))
    return (f_val + μ · l1norm)
```

Merit function: $\phi(x) = f(x) + \mu \|g(x)\|_1$

Algorithm 3.1 (Backtracking Line Search).

Choose $\bar{\alpha} > 0, \rho \in (0,1), c \in (0,1)$; Set $\alpha \leftarrow \bar{\alpha}$;

repeat until $\phi(x_k + \alpha_k p_k) \geq \phi(x_k) + \eta \alpha_k (\nabla f_k^T p_k - \mu \|g(x_k)\|_1)$

$\alpha \leftarrow \rho\alpha$;

end (repeat)

Terminate with $\alpha_k = \alpha$.

If $g(x) = \begin{bmatrix} g(x_1) \\ g(x_2) \end{bmatrix}$ & $g(x_1) = 4$, $g(x_2) = -5$

Then $\|g(x)\|_1 = |4| + |-5| = 9$

& if $f(x) = 10$ & $\mu = 1$

The $\phi(x) = 10 + 1 \times 9 = 19$

Now lets see the case of SQP in non linear optimal control problems:

Given;

$$\min_{x_1, \dots x_T, u_1 \dots u_{T-1}} \sum_{t=0}^{T-1} \left(l_t(x_t, u_t) \right) + l_T(x_T)$$

Terminal cost doesn't need control input u

S.T. $x_{t+1} = f(x_t, u_t)$

It can be expressed as the following QP:

$$\min_{\Delta x, \Delta u} \sum_{k=0}^{T-1} \left(\frac{1}{2} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix}^T \begin{bmatrix} Q_k & S_k^T \\ S_k & R_k \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + \begin{bmatrix} q_k \\ r_k \end{bmatrix}^T \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} \right) + \frac{1}{2} \Delta x_T^T Q_T \Delta x_T + \Delta x_T^T q_T$$

ST $x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + r_{k+1}$

A_k, B_k: solely depend on dynamics of system

ie; $f(x_k, u_k)$

where; $Q_k = (\nabla^2_{xx} \mathcal{L}_k + \lambda^T_{k+1} \nabla^2_{xx} f_k - \mu^T_{k+1} \nabla^2_{xx} c_k)$, $(x_k^{(n)}, u_k^{(n)})$

$S_k = (\nabla^2_{xu} \mathcal{L}_k + \lambda^T_{k+1} \nabla^2_{xu} f_k - \mu^T_{k+1} \nabla^2_{xu} c_k)$, $(x_k^{(n)}, u_k^{(n)})$

$R_k = (\nabla^2_{uu} \mathcal{L}_k + \lambda^T_{k+1} \nabla^2_{uu} f_k - \mu^T_{k+1} \nabla^2_{uu} c_k)$, $(x_k^{(n)}, u_k^{(n)})$

$q_k = (\nabla_x \mathcal{L}_k)$, $(x_k^{(n)}, u_k^{(n)})$

$r_k = (\nabla_u \mathcal{L}_k)$, $(x_k^{(n)}, u_k^{(n)})$

$Q_T = (\nabla^2_{xx} \mathcal{L}_T - \mu_T^T \nabla^2_{xx} c_T)$, $(x_k^{[n]})$

$q_T = (\nabla_x \mathcal{L}_T)$, $(x_k^{[n]})$

$A_k = \nabla_x f_k$, $(x_k^{(n)}, u_k^{(n)})$ $B_k = \nabla_u f_k$, $(x_k^{(n)}, u_k^{(n)})$

This QP can be solved by RICCATI RECURSION

$V_T = Q_T$ $v_T = q_T$

$$h_k = r_k + B_k^\top (v_{k+1} + V_{k+1} \gamma_{k+1})$$

(r_k: linear control penalty)

$$G_k = S_k^\top + B_n^\top V_{k+1} A_k$$

$$H_k = R_k + B_k^\top V_{k+1} B_k$$

$$V_k = Q_k + A_k^\top V_{k+1} A_k - K_k^\top H_k K_k$$

$$v_k = q_k + K_k^\top r_k + (A_k + K_k B_k)^\top (v_{k+1} + V_{k+1} \gamma_{k+1})$$

$K_k = -H_k^{-1} G_k$

$k_k = -H_k^{-1} h_k$