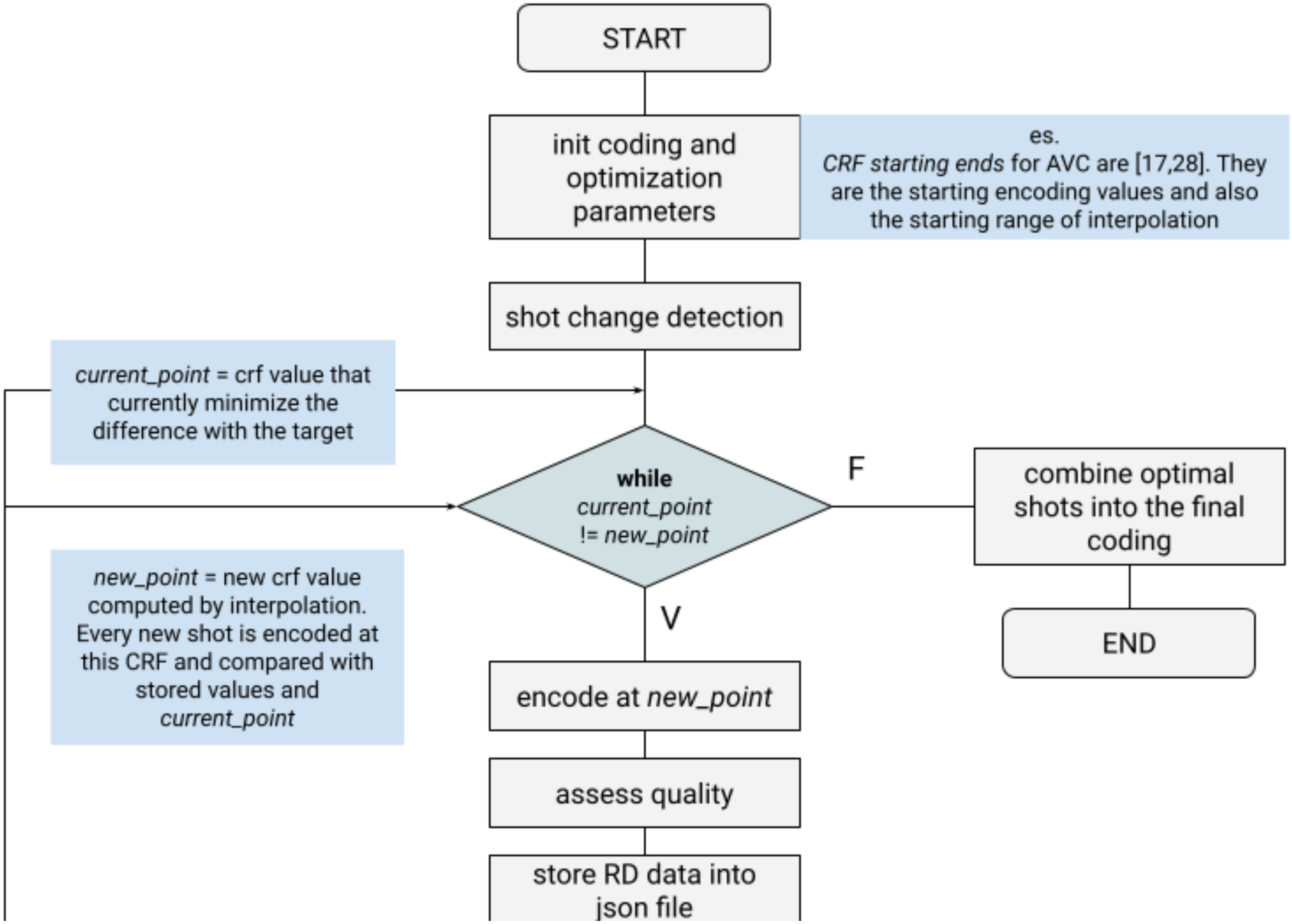
# Dynamic optimization

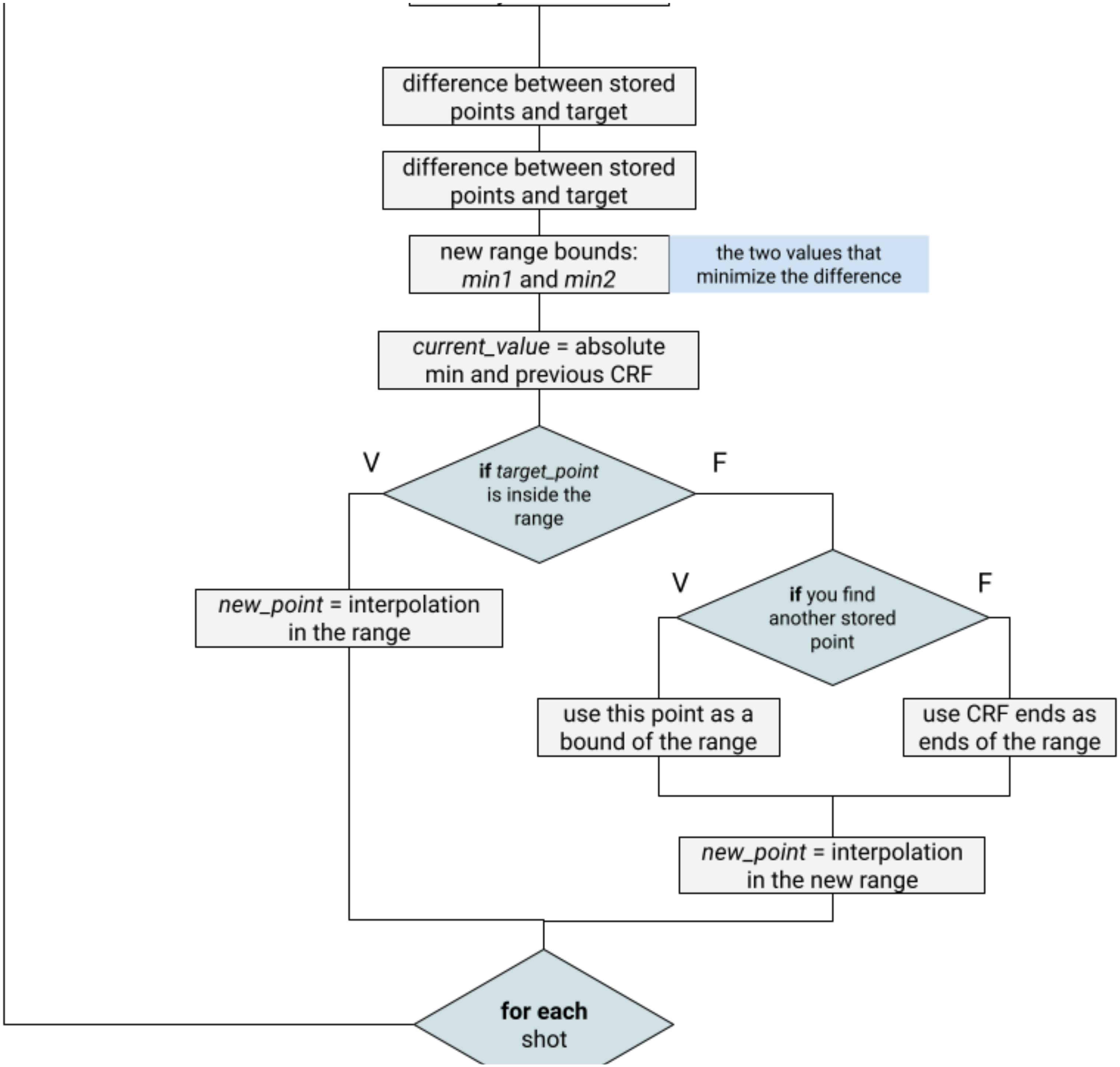
## Intro

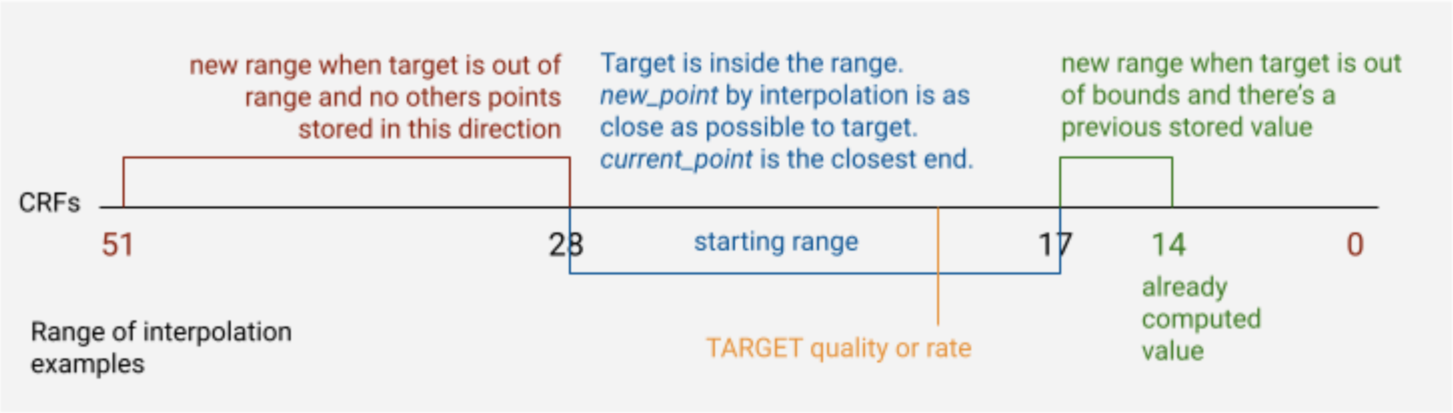

START
init coding and optimization parameters
shot change detection
while *current_point* != *new_point*
F
V
combine optimal shots into the final coding
END
encode at *new_point*
assess quality
store RD data into json file

es.
*CRF starting ends* for AVC are [17,28]. They are the starting encoding values and also the starting range of interpolation

*current_point* = crf value that currently minimize the difference with the target

*new_point* = new crf value computed by interpolation. Every new shot is encoded at this CRF and compared with stored values and *current_point*


difference between stored points and target
difference between stored points and target
new range bounds: *min1* and *min2*
the two values that minimize the difference
*current_value* = absolute min and previous CRF
V
**if** *target_point* is inside the range
F
*new_point* = interpolation in the range
V
**if** you find another stored point
F
use this point as a bound of the range
use CRF ends as ends of the range
*new_point* = interpolation in the new range
**for each** shot


new range when target is out of range and no others points stored in this direction
Target is inside the range. new_point by interpolation is as close as possible to target. current_point is the closest end.
new range when target is out of bounds and there's a previous stored value
CRFs
51
28
starting range
17
14
0
already computed value
Range of interpolation examples
TARGET quality or rate

# Init

In [2]:

```python
# imports
import os #to access system folders
import subprocess #to access ffmpeg in the system
import numpy as np #easy vector operations
import math #for operation with infinite
from scipy.optimize import curve_fit #fittin of the curve
import json #to handle json files
import matplotlib.pyplot as pl #to display plots

#constants
PARAM_AVC = {"crfs": 52, "starting_range": [17,28], "lib": "libx264", "container": "mp4", "add_
PARAM_HEVC = {"crfs": 52, "starting_range": [22,33], "lib": "libx265", "container": "mp4", "add_
PARAM_VP9 = {"crfs": 64, "starting_range": [15,35], "lib": "libvpx-vp9", "container": "webm", "a

#variables
input_type = "yuv" #valules:"yuv", "y4m", "seq"
```

```python
codec = "avc" #values: "avc", "hevc", "vp9", "av1", "vvc"

raw_width = 1920 #opz. only for yuv and seq
raw_height = 1080 #opz. only for yuv and seq
raw_fps = 29.97 #opz. only for yuv and seq

source_name = "testscene01"
source_path = "test_vids/srcRAW_FullHD/" + source_name + "." + input_type
ref_path = "test_vids/tempRAW_refs/" #raw files for each shot
dist_path = "test_vids/temp_encoded/" #encoded files for each shot
#assessment files path
tm_file = "rd_results/template.json"
rd_file = "rd_results/" + source_name + ".json"
vmaf_logs = "rd_results/vmaf_logs"

current_point = None #the current optimum point
new_point = 0 #new value to compare with current_point
#all computed points, by row: crf, bitrate, vmaf, psnr
res_matrix = {"crf": None, "bitrate": None, "vmaf": None, "psnr": None}

flag_target = True #values True (quality) or False (bitrate)
quality_metric = "vmaf" #values "vmaf", "psnr", "ssim", "mssim"
#TODO implement more quality metrics
target_bitrate = [12000000]
target_quality = [96]

print("init done")
```

```
init done
```

## Input and shot change detection

In [3]:

```python
def init_res_matrix(x):
    res_matrix["crf"] = np.arange(0,x,1).tolist()
    inf_matrix = np.zeros(x) #infinity to avoid considering zero as a point
    inf_matrix[inf_matrix == 0] = math.inf
    res_matrix["bitrate"] = inf_matrix.tolist()
    res_matrix["vmaf"] = inf_matrix.tolist()
    res_matrix["psnr"] = inf_matrix.tolist()
```

An empty json structure is generated to be filled and to store computed values

In [4]:

```
struct_points = [] #structure of target points for json file
struct_shots = [] #structure of shots for json file
num_scenes = 3 #TODO: shot change detection

#TODO: support different input raw files
if input_type == "yuv":
    print("yuv input")
elif input_type == "y4m":
    print("y4m input")
else:
    print("No such an input type")
    exit()

#TODO: create folers inside tempENCODED with indexes

#init values based on the selected output codec
if codec == "avc":
    s_cod = PARAM_AVC
    init_res_matrix(PARAM_AVC["crfs"])
elif codec == "hevc":
    s_cod = PARAM_HEVC
    init_res_matrix(PARAM_HEVC["crfs"])
elif codec == "vp9":
    s_cod = PARAM_VP9
    init_res_matrix(PARAM_VP9["crfs"])
else:
    print("No such an codec")
    exit()

min_range_crf = s_cod["starting_range"][0]
max_range_crf = s_cod["starting_range"][1]

#if not os.path.isfile(rd_file):
    #TODO: duplicate file and rename
with open(tm_file, 'r') as f:
    o_data = json.load(f)

    #add source name and results matrix
    o_data["content"] = source_name
    o_data["versions"][0]["codec"] = codec
    o_data["versions"][0]["width"] = raw_width
    o_data["versions"][0]["height"] = raw_height
    o_data["versions"][0]["fps"] = raw_fps
    o_data["versions"][0]["shots"][0]["assessment"] = res_matrix

    #add emplty target points
    base_point = o_data["versions"][0]["shots"][0]["opt_points"][0]
    y = lambda x: target_quality if x else target_bitrate
    for i in range(0, len(y(flag_target))):
        base_point["target"] = y(flag_target)[i]
        struct_points.append(base_point.copy())
    o_data["versions"][0]["shots"][0]["opt_points"] = struct_points

    #add empty shots
    base_shot = o_data["versions"][0]["shots"][0]
    for i in range(0, num_scenes):
        base_shot["index"] = i #assign index to shots in json file
        struct_shots.append(base_shot.copy())
    o_data["versions"][0]["shots"] = struct_shots

with open(rd_file, 'w') as w:
    json.dump(o_data, w, separators=(',',': '))
```

```
yuv input
```

## Optimization

Find the shot encoded to a certain crf that has the closest quality or rate to the target

In [5]:

```
#store the quality and rate results for each shot at each encoded crf
def save_results(index, crf, bitrate, vmaf, psnr):
    with open(rd_file, 'r') as f:
        o_data = json.load(f)
        o_data["versions"][0]["shots"][index]["assessment"]["crf"][crf] = crf
        o_data["versions"][0]["shots"][index]["assessment"]["bitrate"][crf] = bitrate
        o_data["versions"][0]["shots"][index]["assessment"]["vmaf"][crf] = vmaf
        o_data["versions"][0]["shots"][index]["assessment"]["psnr"][crf] = psnr
    with open(rd_file, 'w') as w:
        json.dump(o_data, w, separators=(',',': '))
    print("values for -shot" + str(index) + " -crf" + str(crf) + " saved")
    print("-rate" + str(bitrate) + " -vmaf" + str(vmaf))
    return o_data["versions"][0]["shots"][index]["assessment"]

#linear interpolation of the target and the weight alpha between sx and dx
def interpolate(mat, sx, dx):
    alpha = (mat[target_name][sx] - target) / (mat[target_name][sx] - mat[target_name][dx])
    new_point = round(mat["crf"][sx] - alpha * (mat["crf"][sx] - mat["crf"][dx]))
    print("new -crf" + str(new_point))
    return new_point
```

In [6]:

```
shot_index = 0
point_index = 0

for shot in sorted(os.listdir(ref_path)): #for each shot
    print("init computing -scene" + str(shot_index))
    while not current_point == new_point: #if no convergence
        if point_index == 0: #if there are no points to compare
            new_point = max_range_crf #encode at the upper value iof the starting range

        #encoding
        add_info = s_cod["add_param"]
        lib = s_cod["lib"]
        out = dist_path + str(shot_index) + "/" + str(new_point) + "_" + codec.upper() + \
                "." + s_cod["container"]
        enc = f"ffmpeg -f rawvideo -video_size {raw_width}x{raw_height} \
            -r {raw_fps} -pixel_format yuv420p -i {ref_path+shot} -c:v {lib} \
            -crf {new_point} {add_info} {out}"
        subprocess.call(enc, shell=True)

        #quality assessment
        c_vmaf = f"ffmpeg -f rawvideo -r {raw_fps} -video_size {raw_width}x{raw_height} -i {ref
            -i {out} \
            -lavfi \"[0:v]setpts=PTS-STARTPTS[ref];\
                     [1:v]scale={raw_width}x{raw_height}:flags=bicubic, setpts=PTS-STARTPTS[dis
                     [dist][ref]libvmaf=feature=name=psnr:log_path={vmaf_logs}:log_fmt=json\" \
            -f null -" #|name=float_ssim|name=float_ms_ssim to compute the other metrics
        subprocess.call(c_vmaf, shell=True)

        #extract quality and rate values
        with open(vmaf_logs, 'r') as r:
            i_data = json.load(r)
        vmaf = i_data["pooled_metrics"]["vmaf"]["mean"]
        psnr = (6*i_data["pooled_metrics"]["psnr_y"]["mean"] + \
                i_data["pooled_metrics"]["psnr_cb"]["mean"] + i_data["pooled_metrics"]["psnr_cr
        info = "ffprobe -v error -select_streams v:0 -show_entries format:stream -print_format
        cout = subprocess.run(info.split(), stdout=subprocess.PIPE, stderr=subprocess.STDOUT).s
        dict = json.loads(cout)
        bitrate = int(dict['format']['bit_rate'])

        #TODO: results must be weighted based on duration
        res_matrix = save_results(shot_index, new_point, bitrate, vmaf, psnr)

        if flag_target:
            target_name = "vmaf"
            target = target_quality[0] #TODO: support more targets
        else:
            target_name = "bitrate"
            target = target_bitrate[0]

        if point_index == 0: #if there are no points to compare (first loop)
            current_point = max_range_crf #the current optimal point is the first one
            new_point = min_range_crf #in the next loop encode at the lower end of the starting
        else:
            #element-wise difference between the metric and its target value
            difference = np.asarray(abs(np.asarray(res_matrix[target_name]) - target))
            #the minimum difference = the element with the index closer to the target
            i_first_min = np.argmin(difference)
            nd_diff = difference.copy()
            nd_diff[i_first_min] = np.inf #replace the minimum with inf
            i_second_min = np.argmin(nd_diff) #find the second minimum
            current_point = i_first_min #the index of the point closer to the target
            #swap the values of the two ends if the lower end is bigger than the upper end
            if(res_matrix[target_name][i_first_min] > res_matrix[target_name][i_second_min]):
                sx_end = res_matrix[target_name][i_first_min]
                dx_end = res_matrix[target_name][i_second_min]
                i_sx_end = i_first_min
                i_dx_end = i_second_min
            else:
                sx_end = res_matrix[target_name][i_second_min]
                dx_end = res_matrix[target_name][i_first_min]
                i_sx_end = i_second_min
                i_dx_end = i_first_min

            if target < sx_end and target > dx_end: #if the target is in the range
```

```
                new_point = interpolate(res_matrix, i_sx_end, i_dx_end)
            else: #if the target is out of the range
                if target > sx_end: #if the target is out of the range in the left side
                    new_point = 0 #if no other points in this direction had been stored encode
                    i = i_sx_end - 1
                    while new_point == 0 or not i == 0:
                        #the first point you find is the new lower end of the range
                        if not res_matrix[target_name][i] == math.inf:
                            new_point = interpolate(res_matrix, i, i_sx_end)
                        i -= 1
                else: #if the target is out of the range in the right side
                    #if no other points in this direction had been stored, encode at the max cr
                    new_point = s_cod["crfs"]
                    i = i_dx_end + 1
                    while new_point == 0 or not i == s_cod["crfs"]:
                        #the first point you find is the new upper end of the range
                        if not res_matrix[target_name][i] == math.inf:
                            new_point = interpolate(res_matrix, i_dx_end, i)
                        i += 1
        point_index += 1

    print("Opt point: " + str(current_point))
    current_point = None
    new_point = 0
    shot_index += 1
    point_index = 0
```

```
init computing scene0
```

```
ffmpeg version N-106635-g83e1a1de88 Copyright (c) 2000-2022 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.4.0-1ubuntu1~20.04.1)
```

# TODO: Encode opt video

Put together all the individually encoded shots

In [ ]:

```

```

# TODO: Curve fitting

When the upper search has tested 3 points, given these 3 RQ points, discover the polynomian or logarithmic function that describes their trend. Repeat this when a new point is computed. Measure the error between the approximation and the actual implementation (lagrangian search above) and assess whether and when it may be useful to speed up the search process, by reducing the number of test to encode before the optimum.

In [ ]:

```

```