Compressione di immagini tramite la DCT

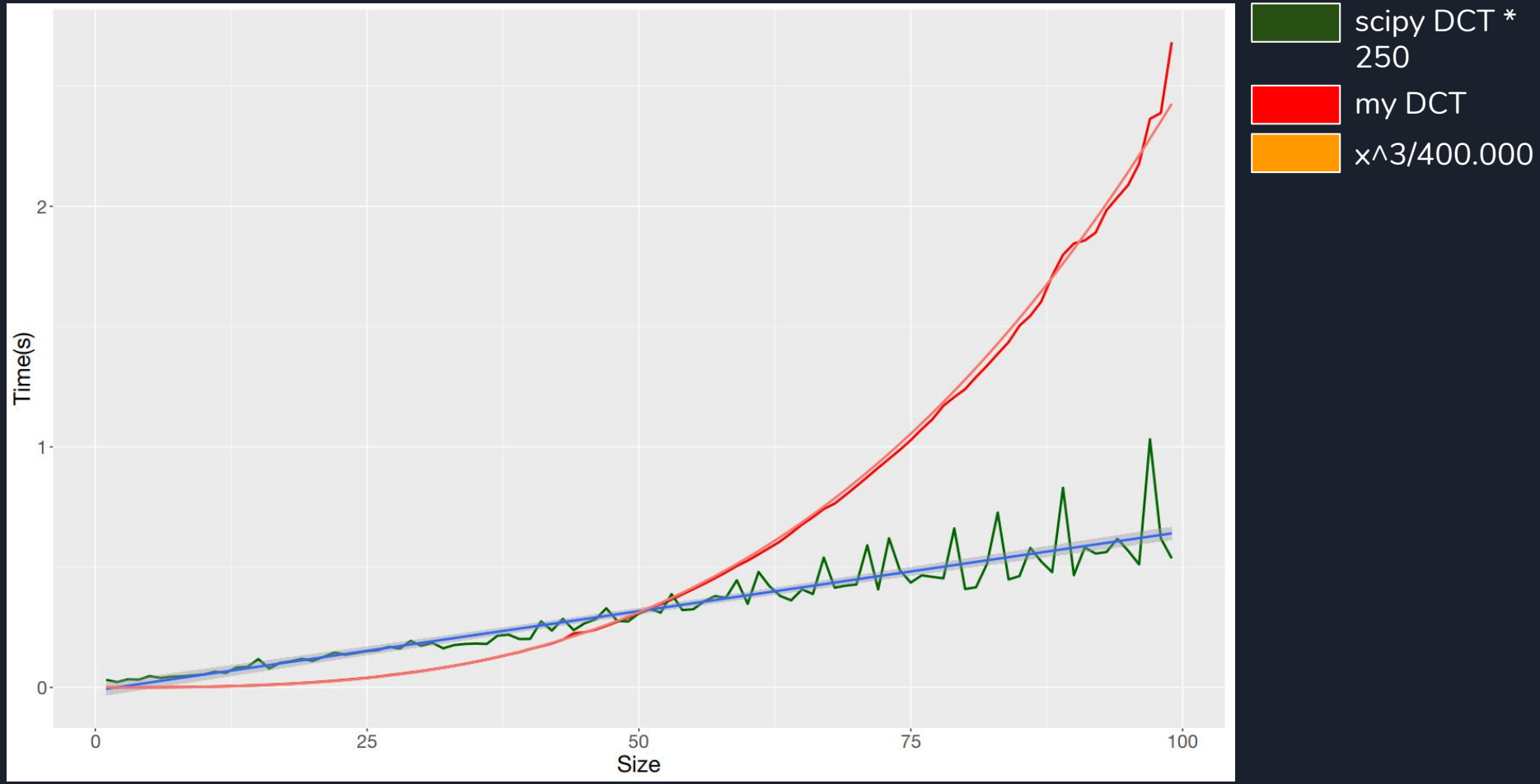
Antonio Vivace, Febbraio 2019

Implementazione casalinga di DCT2

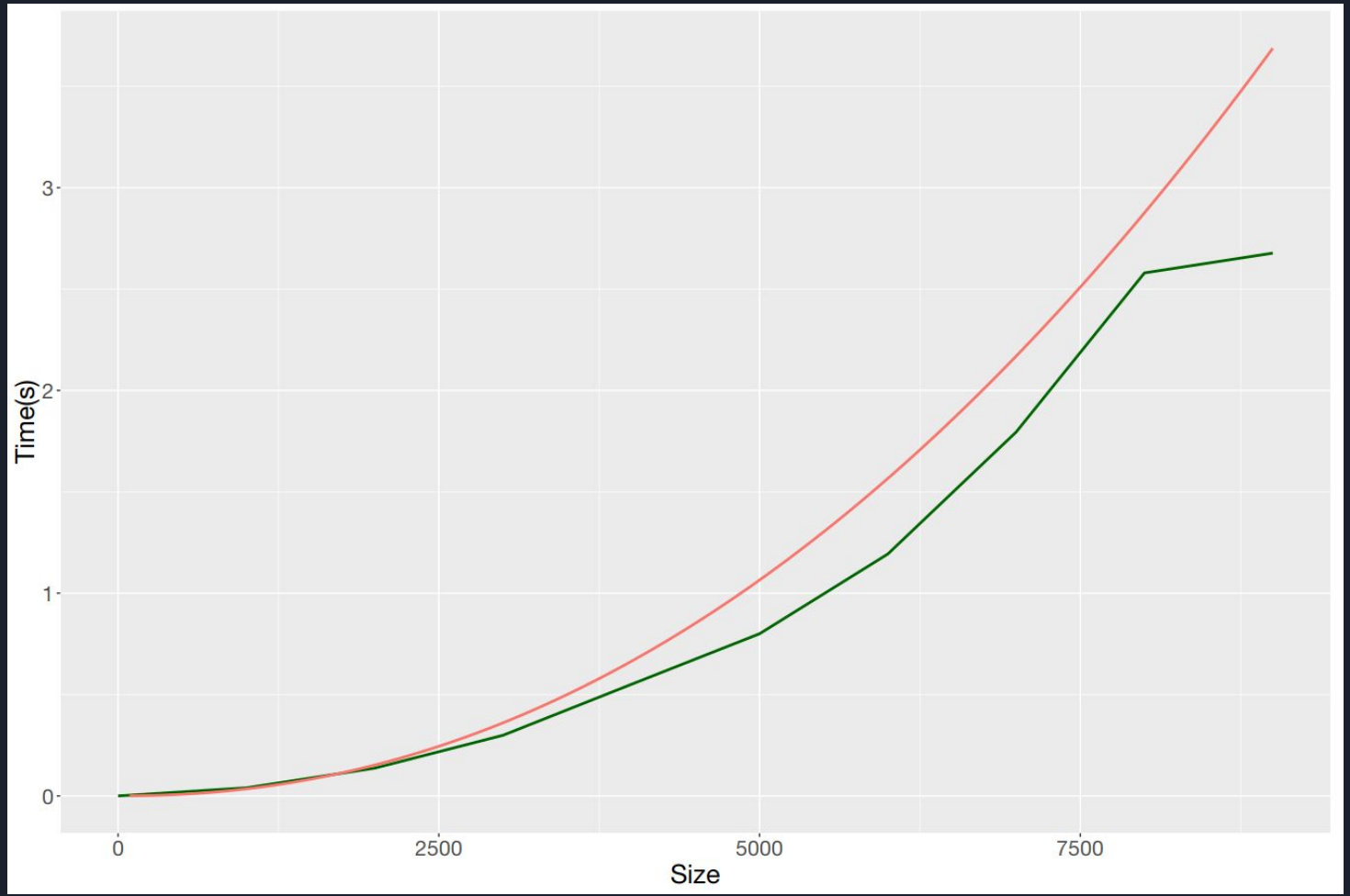
```
# My DCT implementation
def mydct2_mono(a):
    #           N-1
    # y[k] = 2* sum x[n]*cos(pi*k*(2n+1)/(2*N)), 0 <= k < N.
    #           n=0
    n = len(a)
    output = zeros(n)

    for k in range(0, n):
        for i in range(0, n):
            output[k] += a[i]*cos(pi*k*(2*i+1)/(2*n))
        output[k] *= 2
        # If norm='ortho', y[k] is multiplied by a scaling factor f:
        #  f = sqrt(1/(4*N)) if k = 0,
        #  f = sqrt(1/(2*N)) otherwise.
        if k==0:
            output[k] *= sqrt(1/(4*n))
        else:
            output[k] *= sqrt(1/(2*n))
    return output
```

```
def my_dct2(a):
    size1=a.shape[0]
    size2=a.shape[1]
    output = empty([size1, size2])

    # DCT2 (DCT by row and then by column)
    for i in range(0,size1):
        output[i] = mydct2_mono(a[i])

    # The [:, n] notation gives access the n-th column
    for i in range(0,size2):
        output[:, i] = mydct2_mono(output[:, i])

    return output
```

Confronto dei tempi di esecuzione



Velocità di scipy.fftpack.dct



Software Overview

dct.py wrap di DCT2 tramite DCT2(FTT) di scipy (riga e colonne) e implementazione casalinga di DCT2. Esposizione come metodi della classe DCT.

graph.r genera i grafici sulle performance di DCT di scipy contro l'implementazione vanilla/naive in Python casalinga

benchmark.py genera array quadrati di dimensione n crescente e applica la DCT2 di SciPy e quella casalinga, salvandone i risultati in file formattati CSV

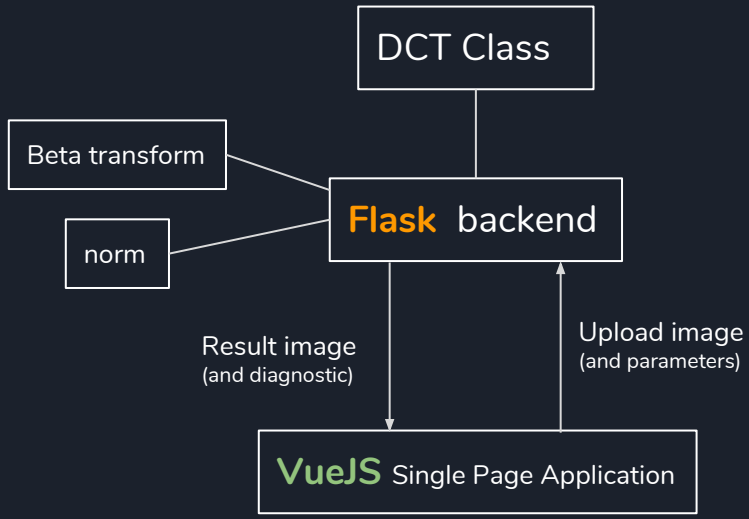
test.py applica test sulle due implementazioni. In particolare, controlla:

- i risultati noti dalla consegna (array completo e DCT monodimensionale);
- idct(dct(x)) = x;
- my_dct(x) = dct(x),

beta.py funzione di trasformazione beta dei c(j,k) con j+k >= d, funzione di arrotondamento e normalizzazione dei valori nei range 0..255.

serverweb.py Server Web in Flask. Espone una rotta RESTful che accetta l'upload di un immagine e i parametri d e beta. Deserializza l'immagine, ne crea un array numpy e applica il workflow richiesto. Fa uso delle classi **dct.py** (metodi SciPy DCT2 e IDCT2), **beta.py** (metodi norm e beta)

Software Overview - Backend



Expose input and output image, input controls for parameters

```
@app.route('/image', methods=["POST"])
def image():

    bytestream = request.files["sourceImage"].read()

    d = int(request.form["d"])
    beta = float(request.form["beta"])


    img = Image.open(io.BytesIO(bytestream))
    f = np.asarray(img, dtype="float")

    c = dct2(f)

    apply_beta_transform(d, beta, c)

    ff = idct2(c).astype(np.uint8)
    norm(ff)

    img = Image.fromarray(ff, mode='L')

    return jsonify(image="data:image/bmp;base64
                  "+pil_to_base64(img).decode('utf8'))
```

Software Overview - Frontend

Applicazione reattiva in VueJS che fa l'upload dell'immagine selezionata al server web esposto da Python Flask.
Visualizza l'immagine selezionata a sinistra, in attesa che vengano selezionati valori di d e beta.

Permette di scegliere l'immagine da una libreria di BMP grayscale 8 preimpostata o di farne l'upload da locale

Fa controllare due input per inserire i valori di **d** e **beta**. **d** è uno slider con min=0 e max=n+m-2, automaticamente settati alla selezione dell'immagine.

Manda l'immagine al backend, che ne applica le trasformazioni e ne restituisce l'immagine risultante, che viene deserializzata, convertita in base64 e rivisualizzata a schermo, di fianco.

Una volta ricevuta l'immagine, ne stampa delle informazioni diagnostiche come il tempo di computazione per ogni fase del processo.

Frontend UI

VueJS, reattività, SPA

Axios per il consumo di rotte

FileReader per le informazioni prima dell'upload dell'immagine (width, height per limitare d)

Vuetify framework CSS