Sorbonne Université

Advanced Numerical Algorithms

Practical 3 : - Fast Fourier Transform

Anatole Vercelloni

Teacher : Stef Graillat

avril 2023

Table des matières

Exercise 4

1.

```
function f = FFT(a)
    f = a;
    n = length(a);
    w = exp(2*1i*pi/n);
    if (n == 1)
        return ;
    end
    b = a(1:2:n);
    c = a(2:2:n);
    f1 = FFT(b);
    f2 = FFT(c);
    rm = rem(n,2);
    q = (n-rm)/2;
    r = 1:n;
    for j = 1:q
        r(j) = f1(j) + power(w, j)*f2(j);
        r(j+ n/2) = f1(j) - power(w, j)*f2(j);
    f = r;
    end
end
```

FIGURE 1 – recursive fft algorithm

2.

```
function A=FFT_it(a)
    n = length(a);
    A = 1:n;
    d = log2(n);
    for k = 1: n
        A(bitrevorder(k)) = a(k);
    end
    for s = 1:d
      m = power(2, s);
      wm = exp(-2*pi*1i/m);
      for k = 0: m: n -1
          w = 1;
          for j = 0 : m/2 - 1
              t = w* A(k + j + 1 + m/2);
              u = A(k + j + 1);
              A(k + j + 1) = u + t;
              A(k + j + 1 + m/2) = u - t;
              w = w*wm;
          end
      end
  end
```

FIGURE 2 – iterative fft algorithm

I tested these two versions of the fft on the examples that we saw in tutorials and the results was correct.

3.

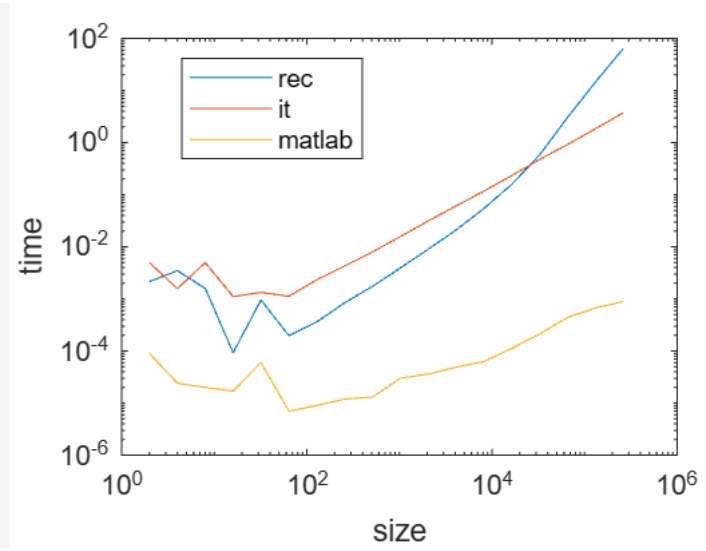

FIGURE 3 – iterative fft algorithm

We can see that, as expected the matlab implementation is the best one. And, for a size sufficiently high, the iterative version is better than the recursive version.

1 Exercise 5

1. I implemented the image compression algorithm

```
function a = image_compression(X, eps)
    [m,n] = size(X);
    fx = fft2(X);
    cr = 0;
    for i=1:m
        for j=1:n
            if abs(fx(i,j))<eps
                fx(i,j) = 0;
                cr = cr + 1;
            end
        end
    end
    cr = cr/(n*m);
    disp(cr);
    a  = ifft2(fx);
end
```

FIGURE 4 – image compression algorithm

2. I computed the rate of compression which is the number of element removed over the number of element. And I get these results

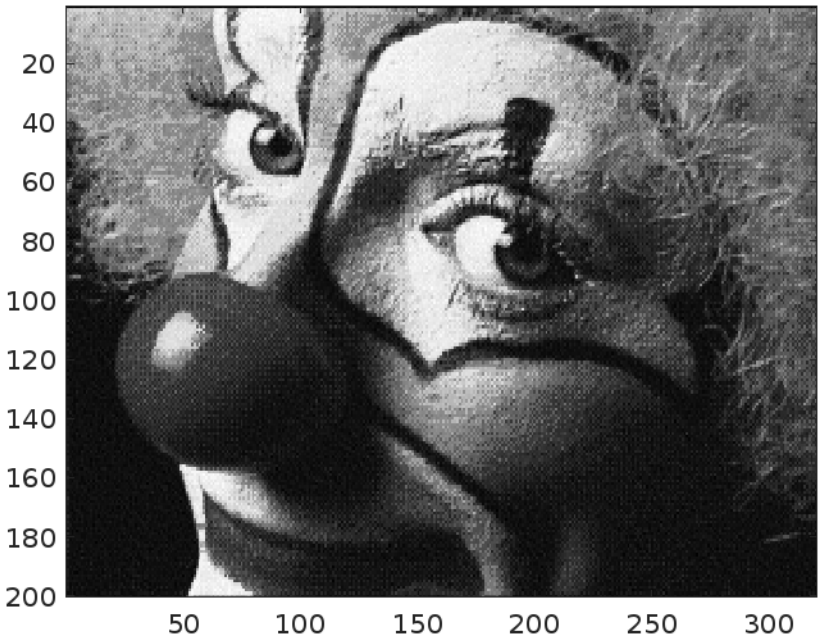

Figure 5 – compression of the clown with a rate of 0.4

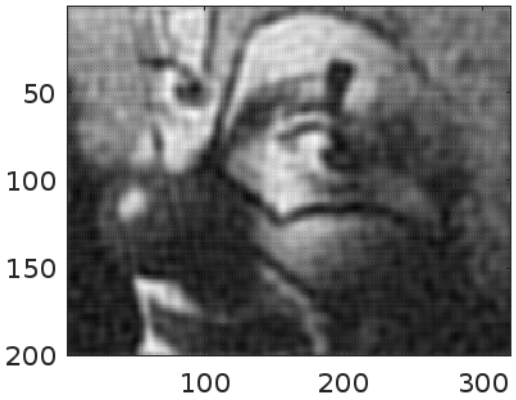

Figure 6 – compression of the clown with a rate of 0.98

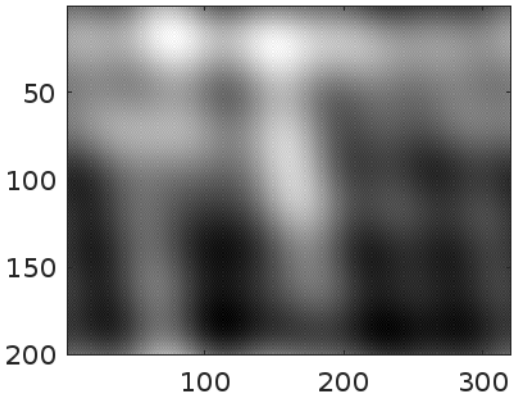

Figure 7 – compression of the clown with a rate of 0.9995

2 Exercise 6

```
function p = polynom_product(p1, p2)
    n1 = length(p1);
    n2 = length(p2);
    d = n1 + n2 + 1;
    fft1 = fft([p1, zeros(1, d - n1)]);
    fft2 = fft([p2, zeros(1, d - n2)]);
    F = fft1.*fft2;
    p = ifft(F);
end
```

FIGURE 8 – polynome product with fft algorithm

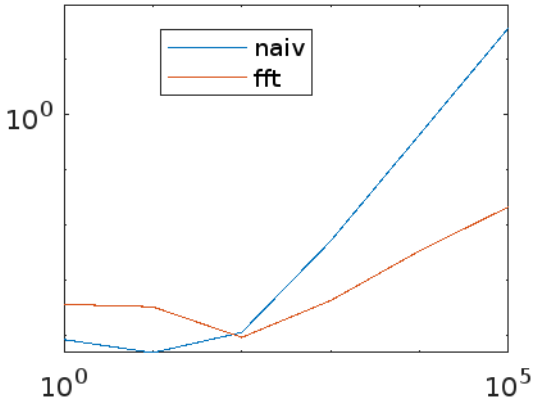

FIGURE 9 – computational time of polynome product algorithm

We can see that the naive one is quadratic whereas the fft is in O(nlog(n))