Number Theoretic Transform with Fast Algorithm

The package includes the following 4 python scripts:

1. `NTT.py`: A class called `fNTT` to do fast Fourier transform and a function called mNTT to do discrete Fourier transform via matrices. Several useful functions are also included.
2. `testNTT.py`: A script to test the speed of using matrix or fast algorithm to do the discrete Fourier transform. One should see big differences when input sequence length is large.
3. `convolve.py`: Functions to do integer convolution.
4. `testconvolve.py`: A script to test the speed of directly convolution or using number theoretic transform with proper `fNTT` class.

We give a simple guide of each file to introduce how to use them and explain how they work.

NTT.py

In this file we primarily provide functions and a class to do number theoretic transform by fast algorithm.

Function Tools

`def modmult(a, b, M)`

Do multiplication followed with a mod

Input parameter

- `a`, `b`: integer
- `M`: integer to mod

Output

- `out`: integer, representing value $(a \times b) \bmod M$

`def modpow(alpha, n, M)`

Do exponential in a finite field Z_M. Note that we do the power by square and multiply algorithm, which takes $\mathcal{O}(k)$ time for k

bit integer.

Input parameter

- `alpha`, `n`: integer
- `M`: integer to mod

Output

- `out`: integer, representing value $\text{alpha}^n \bmod M$

`def is_primitive(alpha, N, M)`

Check whether alpha is a primitive root of order N in finite field Z_M. That is, $\mathrm{alpha}^N = 1 \bmod M$ but $\mathrm{alpha}^i \neq 1 \bmod M$ for all $0 < i < N$.

Input parameter

- `alpha`: integer
- `N`: integer of order
- `M`: integer to mod

Output

- `True` or `False`

`def find_primitive(N, M)`

Return a primitive root of order N in finite field Z_M. If not exist, return 0.

Input parameter

- `N`: integer of order
- `M`: integer to mod

Output

- `out`: integer, the primitive root if exist, or 0.

`def findinv(alpha, M)`

Return the inverse of value alpha in finite field Z_M. If not exist, return None.

Input parameter

- `alpha`: integer
- `M`: integer to mod

Output

- `out`: integer, which represents inverse of `alpha` with respect to modular `M`.

`def bit_reverse(x, n)`

If x is an integer, return the bit-reversed value of x with bit length n. If input x is a list, do bit-reverse for all items. The function is used in initialization of fast Fourier transform since the twiddle factor is used in a bit-reversed order.

Input parameter

- `x`: integer or list.
- `n`: expected bit length of input

Output

- `out`: integer or list.

`def mNTT(N, M, alpha = None)`

Construct the number theoretic transform matrices.

Input parameter

- `N`: integer of size of the transform
- `M`: integer to mod

Output

- `A`: a matrix (numpy array) of forward number theoretic transform
- `B`: a matrix (numpy array) of inverse number theoretic transform

`def CT_Butterfly(x_out, x_in, twiddle_factor, M, length)`

Do Cooley-Tukey butterfly operation by splitting the input sequence into first half and last half.

Input parameter

- `x_out`: numpy array, the output sequence
- `x_in`: numpy array, the input sequence. Note the butterfly is operated on the first half and the last half of input sequence.
- `twiddle_factor`: integer to multiply as twiddle factor in Cooley-Tukey butterfly
- `M`: integer to mod
- `length`: Length to do the butterfly, which is also the length of input.

Output

- `out`: numpy array of operation result. Note that the array `x_out` is also stored with the value.

class fNTT

The class to do the number theoretic transform via the fast Fourier transform algorithm.

`def __init__(self, N, M, alpha = None)`

Constructor of the class.

Input parameter

- `N`: integer, size of the transform.
- `M`: integer to mod
- `alpha`: If provided, use it as the primitive root; otherwise the constructor will search one for it.

`def forward(self, x_in)`

Do forward transform.

Input parameter

- `x_in`: numpy array to do the transform. Note that the input length should be the same size of the class (`N` in constructor)

Output

- `out`: numpy array of the transformed result.

`def inverse(self, x_in)`

Do inverse transform.

Input parameter

- `x_in`: numpy array to do the transform. Note that the input length should be the same size of the class (`N` in constructor)

Output

- `out`: numpy array of the transformed result.

testNTT.py

The script tests the correctness and speed of number theoretic transform defined in `NTT.py`. One can input the parameter of their use, or use our example parameter set.

parameter set

- Set 0: size 16, modular 17 with primitive root 3
- Set 1: size 16, modular 8380417 with primitive root 2883726
- Set 2: size 512, modular 8380417 with prmitive root 1753

Note: Our parameter set 2 is referenced from Dilithium, one of cryptographic signature algorithms in NIST Post-Quantum Cryptography Standardization Third Round finalists.

Usage

```
$ python testNTT.py --help
usage: testNTT.py [-h] [--round ROUND] [--parameter_set PARAMETER_SET] [--N N] [--M M] [--alpha ALPHA]

optional arguments:
  -h, --help            show this help message and exit
  --round ROUND         Number of rounds
  --parameter_set PARAMETER_SET
                        Which parameter set to use (0, 1 or 2)
  --N N                 Size of transform
  --M M                 Modular number
  --alpha ALPHA         Primitive root to used
```

Example Use

```
$ python testNTT.py --round 10000 --parameter_set 0
Testin NTT with size 16, modular 17  10000 rounds

100%|██████████████████████████████████████████████████| 10000/10000 [00:01<00:00, 9654.19it/s]
It takes 5.507469177246094e-05 to initialize fNTT
It takes 1.042314052581787 to compute fNTT 10000 rounds

100%|██████████████████████████████████████████████████| 10000/10000 [00:05<00:00, 1873.33it/s]
It takes 0.00041413307189941406 to initialize mNTT
It takes 5.3382837772369385 to compute mNTT 10000 rounds
```

The fNTT denotes doing number theoretic transform by fast algorithm, and mNTT denotes doing transform via matrix multiplications. Each round includes one forward transform and one inverse transform.

convolve.py

In this file we primarily provide a convolution function to do discrete integer convolution. One can use number theoretic transform to speed up the convolution process. Note that the convolution result via number theoretic transform is correct only if the folloing conditions are satisfied:

1. Since the convolution via Fourier transform is circular-convolution, the size parameter of the transform should be at least sum of length of input sequence minus 1.
2. The result sequence value is equivalent to the correct result with modular. In order to make them equal, the element of input sequence should be limited. Concretely, let the transform size N, with modular M, the input elements should be less than value s where $N \times s^2 < M$.

Function Tools

`def pointwise_modmult(X, Y, M)`

Do point-wise multiplication.

Input parameter

- `X, Y`: numpy arrays.
- `M`: integer to mod

Output

- `out`: numpy array of the multiplication result.

`def convolve(x_in, y_in, ntt = None)`

Do discrete integer convolution. If parameter `ntt` is given, the convolution is done by number theoretic transform.

Input parameter

- `x, y`: numpy arrays.
- `ntt`: an `fNTT` class instance. If provided, the convolution is done by number theoretic transform; otherwise, the convolution is performed via direct calculations.

Output

- `out`: numpy array of the convolution result.

testconvolve.py

The script tests the speed of convolution via number theoretic transform defined in `NTT.py` and `convolve.py`. One can input the parameter of their use, or use our example parameter set.

parameter set

- Set 0: size 16, modular 17 with primitive root 3. The element of input is no larger than 1.
- Set 1: size 16, modular 8380417 with primitive root 2883726. The element of input is no larger than 723.
- Set 2: size 512, modular 8380417 with prmitive root 1753. The element of input is no larger than 127.

Note: Our parameter set 2 is referenced from Dilithium, one of cryptographic signature algorithms in NIST Post-Quantum Cryptography Standardization Third Round finalists.

One should also notice that our test input sequence has length only half of the size of the transform. This is because the convolution via NTT is circular-convolution.

Usage

```
$ python testconvolve.py --help
usage: testconvolve.py [-h] [--round ROUND] [--parameter_set PARAMETER_SET]
                       [--N N] [--M M] [--alpha ALPHA]

optional arguments:
  -h, --help            show this help message and exit
  --round ROUND         Number of rounds
  --parameter_set PARAMETER_SET
                        Which parameter set to use (0, 1 or 2)
  --N N                 Size of transform
  --M M                 Modular number
  --alpha ALPHA         Primitive root to used
```

Example Use

```
$ python testconvolve.py --round 10000 --parameter_set 0
Testing convolution with sequence length 8 10000 rounds

100%|██████████████████████████████| 10000/10000 [00:01<00:00, 6385.76it/s]
It takes 1.5731699466705322 to do convolution via NTT 10000 rounds

100%|██████████████████████████████| 10000/10000 [00:00<00:00, 23811.91it/s]
It takes 0.4201653003692627 to do convolution directly 10000 rounds
```

Each round includes doing a convolution with two randomly sampled sequences.

Experimental Results

We list several experimental results tested on 2021 MacBook Pro with Apple M1 Pro chip. All are tested via scripts `testNTT.py` and `testconvolce.py`.

Number Theoretic Transform

Algorithm	Parameter Set	Size N	Modular M	Round	Time(s)
Fast Algorithm	0	16	17	10000	1.04075
Fast Algorithm	1	16	8380417	10000	1.03723
Fast Algorithm	2	512	8380417	1000	5.15212
Matrix Multiplication	0	16	17	10000	5.41552
Matrix Multiplication	1	16	8380417	10000	5.38856
Matrix Multiplication	2	512	8380417	1000	519.817

One can see that increase of mudular number does not affect the transformation speed, and increase of transformation size results in a significant growing up in time. There is a huge difference in whether to use fast algorithm when transformation has a large size.

Convolution

Algorithm	Parameter Set	Input Length	NTT Size N	Modular M	Round	Time(s)
NTT	0	8	16	17	10000	1.57785
NTT	1	8	16	8380417	10000	1.57110
NTT	2	256	512	8380417	10000	75.6988
Direct Calculation	0	8	-	-	10000	0.42288
Direct Calculation	1	8	-	-	10000	0.42292
Direct Calculation	2	256	-	-	1000	301.505

One can see although direct calculation is faster than using NTT when input sequence is short, it is much more efficient to use NTT when the input length grows.