

Testing the NuHAG Gabor Formulas

Nicolae Tecu & Vlad Vicol

July 16, 2004

Below are the formulas from the NuHAG Gabor Formulas collection that we have tested using Matlab routines.

The testing of the below formulas in Matlab has the purpose to prove that the routines implemented (e.g. for \mathcal{F}_s , $V_g f$, $\eta(K)$, etc.) are correct and consistent with each other.

We first have to state several definitions and conventions:

- $\lambda = (t, \nu) \in \mathcal{G} \times \hat{\mathcal{G}}$
- $t(\nu) = \nu(t) = e^{2\pi i t \cdot \nu}$
- $T_t f(x) := f(x - t)$
- $M_\nu f(x) := \nu(x) f(x)$
- $\pi^*(\lambda) = \overline{\nu(t)} \pi(-\lambda)$
- $(\pi \otimes \pi^*)(\lambda) P = \pi(\lambda) P \pi^*(\lambda)$
- $\mathcal{J} := \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}$ and $\mathcal{J}^* = \mathcal{J}^{-1} = \begin{pmatrix} 0 & -I \\ I & 0 \end{pmatrix}$
- we work with ROW vectors (not column) hence matrix multiplication is done from the right, i.e. $x \cdot A = y$, where x, y are row vectors and A is a matrix.
- In Matlab sequences are indexed starting with 1, this element representing the position 0 in $\mathbb{Z}/n\mathbb{Z}$. Thus given a vector a , of length n , $a(-k)$ is given in Matlab by: $a((-k-1) \bmod n) + 1$.

1 Time-Frequency Operators

$$T_t M_\nu = \overline{\nu(t)} M_\nu T_t \quad (1)$$

$$\pi(\lambda) := M_\nu T_t \quad (2)$$

$$\pi(\lambda_1) \pi(\lambda_2) = \overline{\nu_2(t_1)} \pi(\lambda_1 + \lambda_2) \quad (3)$$

$$\pi(\lambda_1) \pi(\lambda_2) = \nu_1(t_2) \overline{\nu_2(t_1)} \pi(\lambda_2) \pi(\lambda_1) \quad (4)$$

$$(\pi \otimes \pi^*)(\lambda_1) (\pi \otimes \pi^*)(\lambda_2) = (\pi \otimes \pi^*)(\lambda_1 + \lambda_2) \quad (5)$$

Pick a value for n (e.g. 144), a random frequency-shift $freq < n$ and a random time-shift $shift < n$.

1. Testing the commutation formulas for Time-Frequency (TF) shifts (1), the implementation of `rotmod` and `modrot`:

MATLAB:

```
compnorm( exp(-2*pi*i*freq*shift/n) * rotmod(x,shift,freq),
modrot(x,freq,shift) )
```

COMMENTS:

- modrot implements $T_t M_\nu$
- rotmod implements $M_\nu T_t$
- compnorm compares the matrices after normalization

2. Testing the definition of $\pi(\lambda)$ (2) and the compatibility of `tfmat` with `rotmod`

MATLAB:

```
M = randc(n); N = tfmat(n,shift,freq); P = rotmod(M,shift,freq);
compnorm(M*N, P)
```

COMMENTS:

- `tfmat` implements $\pi(\lambda)$

3. Testing the composition of TF operators (3)

MATLAB:

```
mat1 = rotmod(rotmod(M,shift2,freq2), shift1,freq1);
mat2 = exp( -2* pi * i * shift1 * freq2 /n) * rotmod(M,
shift1+shift2, freq1+freq2) ;
compnorm(mat1, mat2)
```

COMMENTS: -

4. Testing commutation formula for TF operators (4)

MATLAB:

```
mat1 = rotmod(rotmod(M,shift1,freq1),shift2,freq2);
mat2 = rotmod(rotmod(M,shift2,freq2),shift1,freq1);
mat2 =
mat2*exp(2*pi*i*shift1*freq2/n)*exp(-2*pi*i*shift2*freq1/n);
compnorm(mat1,mat2)
```

COMMENTS: -

5. Testing properties of the TF-conjugation $(\pi \otimes \pi^*)(\lambda)$ (5) and the compatibility of `tfmat` with `rotmod`

MATLAB:

```
mat1 = tfconj(tfconj(M,shift,freq),shift2,freq2);
mat2 = tfconj(M,shift+shift2, freq + freq2);
compnorm(mat1, mat2)
```

COMMENTS:

- `tfconj` implements $(\pi \otimes \pi^*)(\lambda)$

2 Symplectic Fourier Transform

$$\mathcal{F}_s \circ \mathcal{F}_s = \text{id}_{\mathcal{G} \times \hat{\mathcal{G}}} \quad (6)$$

$$\mathcal{F}_s(f \otimes \hat{g}) = g \otimes \hat{f} \quad (7)$$

$$\mathcal{F}_s(a * b) = \mathcal{F}_s(a) \cdot \mathcal{F}_s(b) \quad (8)$$

$$\langle \mathcal{F}_s a, \mathcal{F}_s b \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} = \langle a, b \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} \quad (9)$$

$$\mathcal{F}_s f(\lambda) = \hat{f}(\mathcal{J}^* \lambda) \quad (10)$$

6. Testing whether the symplectic fourier transform \mathcal{F}_s is it's self inverse (6) and the implementation of `ffts`

MATLAB:

```
A = randc(n); compnorm(A ,ffts(ffts(A)))
```

COMMENTS:

- `ffts` implements \mathcal{F}_s

7. Testing (7) and the implementation of `ffts`

MATLAB:

```
f = randc(1,n); g = randc(1,n); gh = fft(g.');
```

```
A = gh*f; B = ffts(A);
```

```
compnorm(B , fft(f.)*g)
```

COMMENTS: -

8. Testing the compatibility of \mathcal{F}_s with convolutions (8)

MATLAB:

```
a = randc(n); b = randc(n);
```

```
compnorm(ffts(iff2(fft2(a).*fft2(b))),ffts(a).*ffts(b));
```

COMMENTS:

- `ifft2(fft2(a).*fft2(b))` represents the convolution $a * b$

9. Testing whether the \mathcal{F}_s is unitary (9)

MATLAB:

```
a = randc(n); b = randc(n);
```

```
fa = ffts(a); fb = ffts(b);
```

```
componrm( inp(a,b), inp(fb,fa) )
```

COMMENTS:

- `inp(a,b) := b(:)'* a(:)` represents the inner product for matrices

10. Testing the relation between the Symplectic Fourier Transform and the cartesian one (2D) (10)

MATLAB:

```
fa = ffts(a); fb = ffts(b);
```

```
check(9) = compnorm( inp(a,b), inp(fa,fb)) ;
```

COMMENTS:

- `inp(a,b) := b(:)'* a(:)` represents the inner product for matrices

3 Spreading Function, Kohn-Nierenberg Symbol & FFTS

$$\langle a, b \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} = \langle \sigma(a), \sigma(b) \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} \quad (11)$$

$$\langle a, b \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} = \langle \eta(a), \eta(b) \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} \quad (12)$$

$$\langle \eta(a), \eta(b) \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} = \langle \sigma(a), \sigma(b) \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} \quad (13)$$

$$\sigma(I) \equiv 1 \quad (14)$$

$$\eta(I) \equiv \delta_0 \quad (15)$$

$$\eta(K) = \mathcal{F}_s[\sigma(K)] \quad (16)$$

$$\eta(\pi(\lambda)) = \delta_\lambda = T_\lambda \delta \quad (17)$$

$$\sigma((\pi \otimes \pi^*)(\lambda)H) = T_\lambda \sigma(H) \quad (18)$$

$$\eta((\pi \otimes \pi^*)(\lambda)H) = M_{\lambda \mathcal{J}^*} \eta(H) \quad (19)$$

$$\sigma(f \otimes g^*)(x, \xi) = f(x) \overline{\hat{g}(\xi) x(\xi)} \quad (20)$$

$$\eta(GH)(\lambda) = (\eta(G) \natural \eta(H))(\lambda) \quad (21)$$

11. Testing (11) - (15)

MATLAB:

```

compnorm(inp(a,b),inp(mat2kns(a),mat2kns(b)))

compnorm(inp(a,b),inp(mat2spr(a),mat2spr(b)))

compnorm(inp(mat2kns(a),mat2kns(b)),inp(mat2spr(a),mat2spr(b)))

compnorm(mat2kns(eye(n)) , ones(n))

compnorm(mat2spr(eye(n)) , unitmat(n,1,1))

```

COMMENTS:

- `mat2kns(a)` implements $\sigma(a)$
- `mat2spr(a)` implements $\eta(a)$
- `unitmat` implements the dirac matrix

12. Testing how Symplectic Fourier Transform maps the K-N symbol to the spreading function (16)

MATLAB:

```

a1 = spr2mat(randc(n));

compnorm( ffts(mat2kns(a1)), mat2spr(a1) )

```

COMMENTS: -

13. Testing (17)

MATLAB:

```

compnorm(mat2spr(tfmat(n,shift, freq)), unitmat(n,freq+1,shift+1))

```

COMMENTS:

- note that the arguments of `tfmat` and `unitmat` are reversed and the additional 1 comes from Matlab's representation of arrays

14. Testing how TF-conjugation of operators influences the K-N symbol (18)

MATLAB:

```

compnorm(mat2kns(tfconj(M,shift, freq)),
rotrc(mat2kns(M),freq,shift))

```

COMMENTS:

- `rotrc` implements T_λ
- again note the reversed order of the arguments of `rotrc` and `tfconj`

15. Testing how TF-conjugation of operators influences the spreading symbol (19)

MATLAB:

```

l1 = floor( rand(1) * (n-1) ) + 1;

l2 = floor( rand(1) * (n-1) ) + 1;

H = randc(n); HA = mat2spr(tfconj(H,l1,l2));

HC = mat2spr(H); HE = purfr2(n,n,n-l1,l2);

compnorm(HA,HE.*HC)

```

COMMENTS:

- the third argument of the `purfr2` above is `n-l1,...` why ?

16. K-N symbol of rank-one projection (20)

MATLAB:

```

f = randc(1,n); g = randc(1,n);

R = zeros(n); gf = fft(g);

for j=1:n;
    for k = 1:n;
        R(k,j) = f(j)*conj(gf(k)*exp(2*pi*i*(k-1)*(j-1)/n));
    end;
end;
compnorm( R , mat2kns(g'* f) );

```

COMMENTS:

- the order of rows and columns of `R` is reversed

17. Testing the implementation of the twisted convolution (21)

MATLAB:

```

G = randc(n); H = randc(n);

compnorm( mat2spr(G*H) , twistcon(mat2spr(G),mat2spr(H)) );

```

COMMENTS:

- `twistcon` implements the twisted convolution of two operators

4 Short-Time Fourier Transform

$$V_g f(\lambda) = \eta(f \otimes g^*)(\lambda) \quad (22)$$

$$\mathcal{F}_s V_g f(x, \xi) = f(x) \overline{\hat{g}(\xi) x(\xi)} \quad (23)$$

$$V_g \bar{f}(x, \xi) = \overline{x(\xi) V_{\bar{g}} g(-x, -\xi)} \quad (24)$$

$$\langle V_b a, V_d c \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} = \overline{\langle a, c \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})}} \langle b, d \rangle_{(\mathbf{S}_0, \mathbf{L}^2, \mathbf{S}'_0)(\mathcal{G} \times \hat{\mathcal{G}})} \quad (25)$$

$$V_g \tilde{f}(x, \xi) = V_{\tilde{g}} f(-x, -\xi) \quad (26)$$

$$V_g \bar{f}(x, \xi) = \overline{V_{\bar{g}} f(x, -\xi)} \quad (27)$$

$$V_g f(x, \xi) = \overline{x(\xi) V_{\hat{g}} \hat{f}(\xi, -x)} \quad (28)$$

$$V_b a \sharp V_d c = \eta[(a \otimes b^*)(c \otimes d^*)] \quad (29)$$

$$V_b a \sharp V_d c = \langle a, d \rangle V_b c \quad (30)$$

18. Testing the action of the spreading function on rank-one operators (22)

MATLAB:

```
g = randc(1,n); f = randc(1,n);
compnorm(mat2spr(g'*f), stft(f,g)) ;
```

COMMENTS:

- `stft(f,g)` represents $V_g f$

19. Testing (23)

MATLAB:

```
f = randc(1,n); g = randc(1,n);
R = zeros(n); gf = fft(g.'.').';

for j = 1:n;
    for k = 1:n;
        R(k,j) = f(j)*conj(gf(k)*exp(2*pi*i*(k-1)*(j-1)/n));
    end;
end;

compnorm(R, ffts(stft(f,g))) ;
```

COMMENTS:

- the order of rows and columns of `R` is reversed

20. Testing (24), (26) and (27)

MATLAB:

```

f = randc(1,n); g = randc(1,n);

STF1 = stft(conj(f),g); STF2 = stft(g,conj(f)); R = zeros(n);

for k = 1:n;
    for j = 1:n;
        R(k,j) = conj(exp(2*pi*i*(k-1)*(j-1)/n) * STF2(mod(-(k-1),n)+1,mod(-(j-1),n)+1));
    end;
end;

compnorm(STF1, R);

ftilde = f(mod(-([1:n]-1),n)+1);
gtilde = g(mod(-([1:n]-1),n)+1);

s1= stft(ftilde,g); s21= stft(f,gtilde); s2 = s21;

for k = 1:n;
    for j = 1:n;
        s2(k,j) = s21(mod(-(k-1),n)+1,mod(-(j-1),n)+1);
    end;
end;

compnorm(s1, s2);

s1 = stft(conj(f),g); s21 = stft(f,conj(g)); s2 = zeros(n);

for j = 1:n;
    s2(j,:) = s21(mod(-(j-1),n)+1,:);
end;

compnorm(s1 , conj(s2));

```

COMMENTS:

- $\text{conj}(f)$ represents \bar{f}
- $\tilde{f}(x) = f(-x)$
- $f(-x)$ in Matlab represents $f(\text{mod}(-(x-1),n)+1)$

21. Testing the fundamental T-F identity (28)

MATLAB:

```

f = randc(1,n); g = randc(1,n);

STF1 = stft(f,g); STF2 = stft(fft(f),fft(g))/n;

STF2new = zeros(n);

for j = 1:n;
    STF2new(:,j) = STF2(mod(-(j-1),n)+1,:).' .* (exp(-2*pi*i*(j-1)*[0:n-1]/n)).';

```

end;

compnorm(STF1,STF2new)

COMMENTS:

- in the definition of STF2 we need to divide by n in order to account for the factor inserted by the two `fft`'s

22. Testing the orthogonality relation for the `stft` (25)

MATLAB:

```
a = randc(1,n); b = randc(1,n); c = randc(1,n); d = randc(1,n);
```

```
f1 = stft(a,b); f2 = stft(c,d);
```

```
z1 = f1(:)' * f2(:); z2 = conj(a*c'); z3 = b*d';
```

```
compnorm(z1/n , z2 * z3);
```

COMMENTS:

- note that the conjugation comes over $\langle a, c \rangle$ and not over $\langle b, d \rangle$

23. Testing the properties of the twisted convolution (29) and (30)

MATLAB:

```
a = randc(1,n); b = randc(1,n);
```

```
c = randc(1,n); d = randc(1,n);
```

```
compnorm(twistcon(stft(a,b),stft(c,d)),a*d'*stft(c,b));
```

```
compnorm(twistcon(stft(a,b),stft(c,d)),mat2spr((b'*a) * (d'*c)));
```

COMMENTS: -

5 Additional Relations

In the following $G_{\mathbf{m}}$ denotes the Gabor multiplier with upper symbol \mathbf{m} , while λ, λ° represent a lattice and its adjoint.

$$\eta(G_{\mathbf{m}}) = \mathcal{F}_s \left(\sum \mathbf{m}_\lambda \delta_\lambda \right) V_{gg} \quad (31)$$

$$\mathcal{F}_s(\Lambda) = \Lambda^\circ \quad (32)$$

24. Testing (31)

MATLAB:

```

g = randc(1,n);
a = 1; b = 1;

W = randc(n,n) .* randc(n,n);
A = W(1:b:n,1:a:n);
GM=gabmulmh(W,g,a,b);

sprGM = mat2spr(GM); fW = ffts(W); fW(1:b:n,1:a:n);

compnorm(sprGM(1:b:n,1:a:n) , fW(1:b:n,1:a:n).*stft(g,g,a,b));

```

COMMENTS:

- The present implementation of identity (31) works only for $a = b = 1$.

25. Testing (32)

MATLAB:

```

xpo = lattp(144,12,16);
xp1 = adjlat3(xpo);

compnorm(xp1,lattp(144,144/16,144/12));

```

COMMENTS:

- `adjlat3` is a routine that computes the adjoint lattice based on the relation (32).