Mathematics of photonic crystals

Tutorial: An introdiuction to INTLAB

Dagmar Roth
(Karlsruhe Institute of Technology)

Intensive Programme / Summer School „Periodic Structures in Applied Mathematics“
Göttingen, August 18 – 31, 2013

Lifelong Learning Programme
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Grant Agreement Reference Number: D-2012-ERA/MOBIP-3-29749-1-6

Tutorial:

An introduction into

INTLAB

For questions: dagmar.roth@kit.edu

Validated computation by INTLAB

INTLAB (INTerval LABoratory)

→ **Toolbox for reliable computing in MATLAB**

http://www.ti3.tu-harburg.de/rump/intlab/

- **Change of rounding mode**
- **Interval arithmetic (real, complex)**
- **Enclosures for standard functions**

Change of rounding mode

Floating point system based on IEEE Standard 754

$F \subset \mathbb{R}$ **: set of floating points**

- **Rounding upwards** $\Delta : \mathbb{R} \to F$

  Rounds to the smallest floating point which is equal to or larger than c.

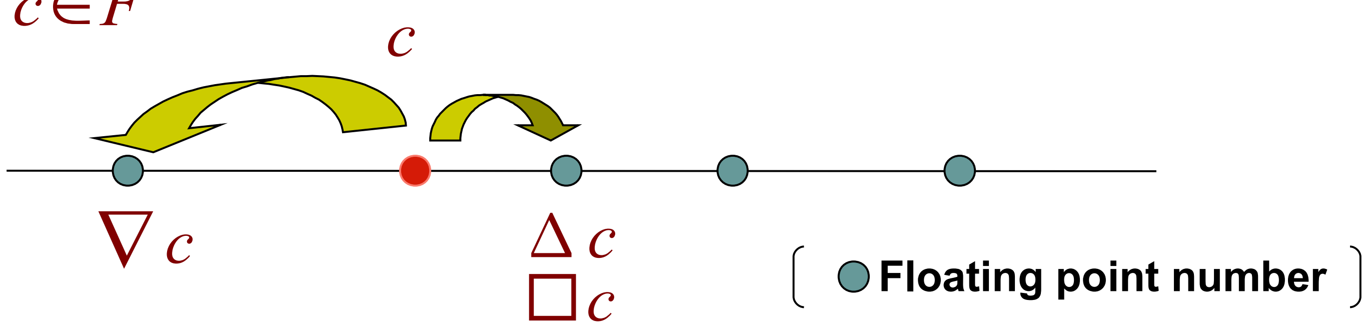
- **Rounding downwards** $\nabla : \mathbb{R} \to F$

  Rounds to the largest floating point which is equal to or smaller than c.

- **Rounding to nearest** $\Box : \mathbb{R} \to F$

  Rounds to the floating point which is the nearest to c.

$c \in F$



Verified computation by change of rounding mode

【Example】

Compute the inner product z

of $x=(x_1,\ldots,x_N),\ y=(y_1,\ldots,y_N)$

Here setround (up) and setround (down) stand for the rounding upwards and downwards respectively.

setround (down);

$$\underline{z}=\sum_{k=1}^{N} x_k y_k;$$

setround (up);

$$\overline{z}=\sum_{k=1}^{N} x_k y_k;$$

→

$$\underline{z}\le z\le \overline{z}$$

is assured in mathematically rigorous sense.

setround function in INTLAB

→ Changes the rounding mode

setround(1) ······· Rounding upwards

setround(-1) ······· Rounding downwards

setround(0) ······· Rounding to nearest

```
>> getround
ans =
    0
```

→ We can check the current rounding mode by “getround”.
（The default mode is Rounding to nearest.）

Remark： If you once changed the rounding mode by the “setround” then the mode is maintained until you change it again.

e.g. Compute $\sum_{i=1}^{100000} 0.1 = 10^4$ with different rounding modes

```
>> setround(0); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   1.000000000001885e+004
```

Rounding to nearest

```
>> setround(1); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   1.000000000003054e+004
```

Rounding upwards

```
>> setround(-1); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   9.999999999947979e+003
```

Rounding downwards

Input of an interval

■ intval

c = intval(x) type cast for x double or intval
c = intval(s) verified conversion for string s

```
>> a=intval(0.1)
intval a =
  0.10000000000000
>> rad(a)
ans =
    0
>> b=intval('0.1')
intval b =
  0.10000000000000
>> rad(b)
ans =
   1.387778780781446e-017
```

generates point interval a, but $0.1 \notin a$

a) 0.1 is rounded (according to rounding mode) to floating-point number $a_f \in F$

b) point interval is constructed which includes a_f

Input via *strings:* generates a real interval b satisfying $0.1 \in b$

※ "rad" means the radius of an interval. (= half of width)

Caution:

interval a = intval(x)
generates a point interval a, which satisfies $x \in a$ if and only if x has an exact representation as floating point number, e.g. x is a rational with denominator being a power of 2 (such as 1/2 or 3/16)

```
>> c=intval(1)/10
intval c =
   0.1000000000000
>> rad(c)
ans =
   1.387778780781446e-017
```

Also possible:
Use standard interval arithmetic for input: generates a real interval c with $0.1 \in c$

Interval vector and Interval matrix

```
>> A=intval([0.1 0.2 0.3 0.4])
intval A =
  0.10000000000000   0.20000000000000   0.30000000000000   0.40000000000000
>> rad(A(1,1))
ans =
    0
>> B=intval(['0.1 0.2 0.3 0.4'])
intval B =
  0.10000000000000
  0.20000000000000
  0.30000000000000
  0.40000000000000
>> B=reshape(B,2,2)
intval B =
  0.10000000000000   0.30000000000000
  0.20000000000000   0.40000000000000
>> rad(B(1,1))
ans =
   1.387778780781446e-017
```

→ **Interval generated by string always leads to a column vector! The desired dimension can be obtained using reshape(B,n,m) (creates matrix with n rows and m columns from input vector/matrix B)**

- **infsup(a,b)** ・・・ **generates an interval which includes [a,b]**

```
>> a=infsup(3,4)
intval a =
[    3.000,    4.000]
```

interval a

```
>> A=[infsup(3,4), infsup(1,2); infsup(0.5,0.75), infsup(0,2.5)]
intval A =
[    3.0000,    4.0000] [    1.0000,    2.0000]
[    0.5000,    0.7501] [    0.0000,    2.5000]
```

interval matrix A

- **midrad(c,w)** ・・・ **generates an interval whose midpoint is c, and radius is w**

```
>> a=midrad(0.5, 1e-3)
intval a =
[    0.4989,    0.5011]
```

interval a

```
>> A=[midrad(0.5,1e-3); midrad(0.25,1e-3)]
intval A =
[    0.4989,    0.5011]
[    0.2489,    0.2511]
```

interval vector A

```
>> B=[midrad(0.25,1e-3), midrad(0.5,1e-3); midrad(0.75,1e-3), midrad(1,1e-3)]
intval B =
[    0.2489,    0.2511] [    0.4989,    0.5011]
[    0.7489,    0.7511] [    0.9989,    1.0011]
```

interval matrix B

Change of display for an interval

In INTLAB there are three ways of output for an interval :

1. added _ e.g : 2.00__
   (_ means uncertainties)

2. [a, b] e.g : [0.1, 0.1001]
   (lower bound is “a”, upper bound is “b”)

3. <a, r> e.g : <1.0, 1e-3>
   (midpoint is “a”, radius is “r”)

The default output is 1.
“intvalinit” or format can change the way of displaying

1	⟶	intvalinit('display_') or format _
2	⟶	intvalinit('displayinfsup') or format infsup
3	⟶	intvalinit('displaymidrad') or format midrad

Example

```
>> intvalinit('display_')
===> Default display of intervals with uncertainty (e.g. 3.14_)
>> pi=midrad(pi,1e-3)
intval pi =
   3.14__

>> intvalinit('displayinfsup')
===> Default display of intervals by infimum/supremum (e.g. [ 3.14 , 3.15 ])
>> pi
intval pi =
[    3.1405,    3.1426]

>> intvalinit('displaymidrad')
===> Default display of intervals by midpoint/radius (e.g. < 3.14 , 0.01 >)
>> pi
intval pi =
<    3.1416,   0.0011>
```

Output in intlab is rigorous, i.e. left <= ans <= right for infsup or $ans \in mid \pm rad$ for midrad

A complex interval is set by the midpoint/radius notation with the midpoint as a complex number.

```
>> format _
>> c=midrad(1+2i,0.01)
intval c =
   1.00_____________ +  2.00_____________i

>> format midrad
>> c
intval c =
<   1.00000000000000 +  2.00000000000000i,  0.01000000000001>

>> format infsup
===> Default display of intervals by infimum/supremum (e.g. [ 3.14 , 3.15 ])
>> c
intval c =
[   0.98999999999998 +  1.98999999999999i,   1.01000000000001 +  2.01000000000001i]
```

midrad(a, r) generates a real interval if “a” is real, so you should use “cintval” if you want to generate a complex interval which has a real midpoint.

```
>> intvalinit('displaymidrad')
===> Default display of intervals by midpoint/radius (e.g. < 3.14 , 0.01 >)
>> c=midrad(0.0, 1e-3)
intval c =
<   0.00000000000000,  0.00100000000001>
>> c=cintval(0.0, 1e-3)
intval c =
  1.0e-002 *
<   0.00000000000000 +  0.00000000000000i,  0.10000000000001>
```

※ A complex interval is always treated in the form of

$$\{z \in \mathbb{C} : |z - a| \le r\}$$

i.e. the midrad representation is used

Interval Arithmetic

$a, b:$ **intervals**

formula	expression in INTLAB
$a+b$	**a + b**
$a-b$	**a - b**
ab	**a * b**
a/b	**a / b**

※ **Elementwise operation works as in MATLAB（a .* b etc.）**

comparison operation → **true: 1 false: 0**

formula	expression in INTLAB
$a < b$	`a < b`
$a \leq b$	`a <= b`
$a > b$	`a > b`
$a \geq b$	`a >= b`
$a = b$	`a == b`
$a \neq b$	`a ~= b`
$a \subseteq b$	`in(a,b)`
$a \subset b$	`in0(a,b)`

Example for “in” and “in0”

```
>> a=infsup(1,3)
intval a =
[   1.00000000000000,   3.00000000000000]
>> b=infsup(0,3)
intval b =
[   0.00000000000000,   3.00000000000000]
>> c=infsup(0,4)
intval c =
[   0.00000000000000,   4.00000000000000]
>> in(a,b)
ans =
     1
>> in0(a,b)
ans =
     0
>> in0(a,c)
ans =
     1
```

Interval trigonometric functions etc. （cf. “help intval”）

```
>> a
intval a =
[   1.00000000000000,   3.00000000000000]
>> sin(a)
intval ans =
[   0.14112000805986,   1.00000000000000]
>> a=infsup(0,pi/2)
intval a =
[   0.00000000000000,   1.57079632679490]
>> sin(a)
intval ans =
[   0.00000000000000,   1.00000000000001]
>> exp(a)
intval ans =
[   1.00000000000000,   4.81047738096536]
>> cosh(a)
intval ans =
[   1.00000000000000,   2.50917847865806]
>> log(a)
intval ans =
[             - Inf,   0.45158270528946]
```

There are several functions for enclosures.

Example 1: verifylss

Enclose the solution of $Ax = b$

```
>> A=rand(5); b=rand(5,1);
>> verifylss(A,b)
intval ans =
[   0.83144502264404,   0.83144502264405]
[   0.70563459485949,   0.70563459485952]
[   1.15370741392371,   1.15370741392373]
[  -1.22407501349193,  -1.22407501349192]
[  -0.82811808977386,  -0.82811808977385]
```

If A is an interval matrix then

A\b

is also available.

```
>> A=midrad(A,1e-4);
>> A\b
intval ans =
[   0.82933211341759,   0.83355793187051]
[   0.70286710112650,   0.70840208859252]
[   1.15008203307291,   1.15733279477453]
[  -1.22949917932440,  -1.21865084765946]
[  -0.83160470085303,  -0.82463147869468]
```

Example 2: verifyeig

Compute a verified eigenpair of $Ax = \lambda x$

```
>> A=wilkinson(9)
A =
    4    1    0    0    0    0    0    0    0
    1    3    1    0    0    0    0    0    0
    0    1    2    1    0    0    0    0    0
    0    0    1    1    1    0    0    0    0
    0    0    0    1    0    1    0    0    0
    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    1    2    1    0
    0    0    0    0    0    0    1    3    1
    0    0    0    0    0    0    0    1    4
```

```
>> [V,D]=eig(A);
>> for i=1:9 disp(D(i,i))
end
  -1.12542241567332
   0.25471875982586
   0.95258421907521
   1.82271708088711
   2.17828473954998
   3.17728291911289
   3.24739647257898
   4.74528124017414
   4.74715698446914
```

computation of approximate eigenpairs

In case that you want to enclose the first eigenpair :

```
>> [L,X]=verifyeig(A,D(1,1),V(:,1))
intval L =
[  -1.12542241567332,  -1.12542241567331]
intval X =
[  -0.00742897533695,  -0.00742897533694]
[   0.03807663671744,   0.03807663671745]
[  -0.14965323529066,  -0.14965323529065]
[   0.42965293943800,   0.42965293943801]
[  -0.76354075315081,  -0.76354075315080]
[   0.42965293943800,   0.42965293943801]
[  -0.14965323529066,  -0.14965323529065]
[   0.03807663671744,   0.03807663671745]
[  -0.00742897533695,  -0.00742897533694]
```

→ **Enclosure of the eigenvalue** (L)

} **Enclosure of the eigenvector** (X)

Exercise (Matrix Eigenvalue Problem)

$p, q : (0,1) \to \mathbb{R}$ **periodic, positive**

$$(1) \quad \begin{cases} -\left(p(x)u'(x)\right)' + q(x)u(x) = \lambda u(x) & \text{in } (0,1) \\ u(1) = e^{i\mu} u(0), \quad u'(1) = e^{i\mu} u'(0) & (\mu \in \mathbb{R}) \end{cases}$$

Weak Formulation

$$a(u,v) := \int_0^1 \left[p(x)u'(x)\overline{v'(x)} + q(x)u(x)\overline{v(x)} \right] dx, \quad u, v \in H^1_{per}(0,1)$$

$$H^1_{per}(0,1) = \left\{ u \in H^1(0,1) : \ u(1) = e^{i\mu} u(0) \right\}$$

Find $(\lambda, u) \in \mathbb{C} \times H^1_{per}(0,1)$ **such that** $a(u,v) = \lambda \langle u, v \rangle_{L^2}$ **for all** $v \in H^1_{per}(0,1)$

Approximate eigenfunction

$$A = \left(A_{nm}\right)_{-N \le n, m \le N}, \qquad B = \left(B_{nm}\right)_{-N \le n, m \le N}$$

$$A_{nm} := a(\phi_n, \phi_m), \qquad B_{nm} := \langle \phi_n, \phi_m \rangle_{L^2} \qquad \left(\phi_n(x) = e^{i(2\pi n + \mu)x}\right)$$

$$A_{nm} = \int_0^1 \left[p(x)\phi_n{}'(x)\overline{\phi_m{}'(x)} + q(x)\phi_n(x)\overline{\phi_m(x)} \right] dx$$

$$B_{nm} = \int_0^1 \phi_n(x)\,\overline{\phi_m(x)}\, dx$$

Matrix Eigenvalue problem:

$$Az = \hat{\lambda} B z$$

$$z = \left(c_n\right)_{-N \le n \le N}$$

$(\hat{\lambda}, z)$ eigenpair ⇒ $u_N(x) := \sum_{n=-N}^{N} \overline{c_n}\phi_n(x)$

is approximate eigenfunction

Using the ansatz

$$u(x)=\sum_{j=-N}^{N}\overline{c_j}\,\phi_j(x)$$

gives

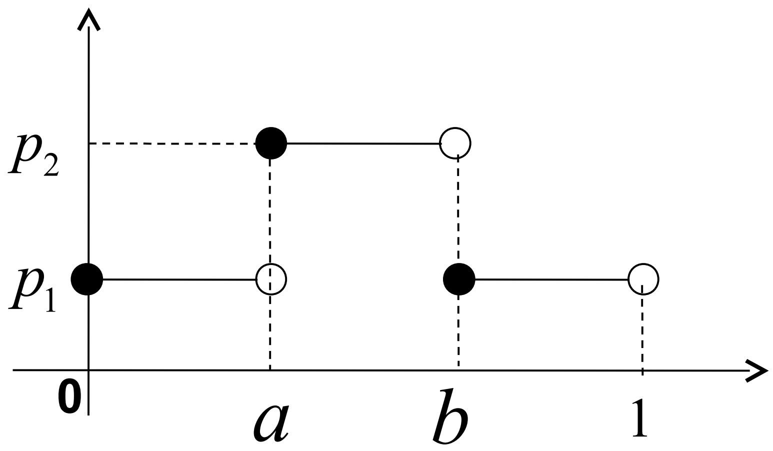
$$a\left(\phi_i,\sum_{j=-N}^{N}\overline{c_j}\,\phi_j\right)=\hat{\lambda}\left\langle\phi_i,\sum_{j=-N}^{N}\overline{c_j}\,\phi_j\right\rangle_{L^2}\qquad i=-N,\ldots,N$$

$$\Leftrightarrow\quad\sum_{j=-N}^{N}c_j\,a(\phi_i,\phi_j)=\hat{\lambda}\sum_{j=-N}^{N}c_j\langle\phi_i,\phi_j\rangle\qquad i=-N,\ldots,N$$

$$\Leftrightarrow\quad A\,z=\hat{\lambda}\,B\,z$$

Example

p : piecewise constant, q : constant



$$A_{nm}=\int_0^1\left[p(x)\phi_n'(x)\overline{\phi_m'(x)}+q(x)\phi_n(x)\overline{\phi_m(x)}\right]dx$$

$$=p_1\int_0^a\phi_n'(x)\overline{\phi_m'(x)}\,dx+p_2\int_a^b\phi_n'(x)\overline{\phi_m'(x)}\,dx$$

$$+p_1\int_b^1\phi_n'(x)\overline{\phi_m'(x)}\,dx+q\int_0^1\phi_n(x)\overline{\phi_m(x)}\,dx$$

$$B_{nm}=\int_0^1\phi_n(x)\,\overline{\phi_m(x)}\,dx \qquad (-N\le n,m\le N)$$

Task 1: a) Calculate (by hand) matrices A and B
b) Implement A and B (no verification at this point)
c) Compute approximate eigenpairs

Use

$$N=50, a=0.3, b=0.6, p_1=p1=1, p_2=p2=2, q=1, \mu=mu=1$$

Remember: [V,E]=eig(A,B);

gives diagonal matrix E containg the eigenvalues of $Az=\hat{\lambda} Bz$
and a matrix V such that A*V=B*V*E, i.e. V(:,i) is eigenvector corresponding to eigenvalue E(i,i)

```
Evec=diag(E);
[ES,I]=sort(Evec);
```

Then V(:,I(k)) gives the eigenvector corresp. to k-th eigenvalue (ordered by magnitude, counted by multiplicity)

```
>> [S,I]=sort([3,2,6,1,2])
S =
    1   2   2   3   6
I =
    4   2   5   1   3
```

$$B_{nn} = 1, \quad B_{nm} = 0 \ \text{ for } \ n \neq m \quad (-N \leq n, m \leq N)$$

$$A_{nn} = (2\pi n + \mu)^2 \left[p_1 (a + 1 - b) + p_2 (b - a) \right] + q \quad (-N \leq n \leq N)$$

$$A_{nm} = \frac{(2\pi n + \mu)(2\pi m + \mu)}{2\pi i (n - m)} (p_1 - p_2) \left(e^{2\pi i (n-m) a} - e^{2\pi i (n-m) b} \right)$$

$$\text{for } \ n \neq m \ \ (-N \leq n, m \leq N)$$

$$\Rightarrow \quad \boxed{Az = \hat{\lambda} z}$$

```
%Eigenvalue computation for 1D interface problem
%-(pu')'+qu=lambda*u
%p=p1 on (0,a) and (b,1),   p=p2 on (a,b)
%q: constant

function interface_eigenvalues

clear
p1=1;  p2=2; a=0.3; b=0.6; mu=1; N=50; q=1;

A=zeros(2*N+1,2*N+1);

for n=-N:N
   A(n+N+1,n+N+1)=(2*pi*n+mu)^2*(p1*(a+1-b)+p2*(b-a))+q;
end

%Initialize A
for n=-N:N-1
   for m=n+1:N
      A(n+N+1,m+N+1)=(2*pi*n+mu)*(2*pi*m+mu)/(2*pi*1i*(n-m))*...
            (p1-p2)*(exp(2*pi*1i*(n-m)*a)-exp(2*pi*1i*(n-m)*b));
      A(m+N+1,n+M+1)=conj(A(n+N+1,m+N+1));
   end
end
```

Let now approximate eigenfunctions be given by

$$u_k(x) := \sum_{n=-N}^{N} \overline{c_n^{(k)}} \phi_n(x), \qquad k = 1, \ldots, M$$

Define

$$A^{\text{new}} = \left(A_{kl}^{\text{new}}\right)_{-N \le k, l \le N}, \qquad B^{\text{new}} = \left(B_{kl}^{\text{new}}\right)_{-N \le k, l \le N}$$

$$A_{kl}^{\text{new}} = a\left(u_k, u_l\right), \qquad B_{kl}^{\text{new}} = \left\langle u_k, u_l \right\rangle_{L^2}$$

⇒

$$A^{\text{new}} z = \hat{\lambda} B^{\text{new}} z$$

Task 2: a) Calculate (by hand) matrices A^{new} and B^{new}
b) Implement A^{new} and B^{new} (interval arithmetic!)
c) Compute enclosures of matrix eigenvalues for

$$A^{\text{new}} z = \hat{\lambda} B^{\text{new}} z$$

Use

$$N = 50, M = 5, a = intval('0.3'), b = intval('0.6'),$$
$$p_1 = p1 = intval(1), p_2 = p2 = intval(2), q = intval(1)$$
$$\mu = mu = intval(1), \pi = piv = acos(intval(-1))$$

Remember: [L,X]=verifyeig(A,E(i,i),V(:,i),B);

gives enclosure L for the i-th eigenvalue of $Az = \hat{\lambda} Bz$
and an enclosure X for the corresponding eigenvector

Results of calculation

$$u_k(x) = \sum_{n=-N}^{N} d_n^{(k)} \phi_n(x), \qquad z^{(k)} = \left(d_n^{(k)}\right)_{-N \le n \le N}, \qquad k = 1, \dots, M$$

$$A_{kl}^{\text{new}} = a\left(u_k, u_l\right) = a\left(\sum_{n=-N}^{N} d_n^{(k)} \phi_n, \sum_{m=-N}^{N} d_m^{(l)} \phi_m \right)$$

$$= \sum_{n,m=-N}^{N} d_n^{(k)} \overline{d_m^{(l)}} \underbrace{a\left(\phi_n, \phi_m\right)}_{A_{nm}} = \left(z^{(k)}\right)^T A\, \overline{z^{(l)}}$$

$$B_{kl}^{\text{new}} = \left\langle u_k, u_l \right\rangle_{L^2} = \ \dots \ = \left(z^{(k)}\right)^T \overline{z^{(l)}}$$

Note: A^new is almost a diagonal matrix with approximate eigenvalues on the diagonal, B^new is almost the unit matrix

```
%Initialize verified A
for n=-N:N-1
    for m=n+1:N
        A(n+N+1,m+N+1)=(2*piv*n+mu)*(2*piv*m+mu)/(2*piv*1i*(n-m))*...
            (p1-p2)*(exp(2*piv*1i*(n-m)*a)-exp(2*piv*1i*(n-m)*b));
        A(m+N+1,n+N+1)=conj(A(n+N+1,m+N+1));
    end
end

for n=1:M
    for m=1:M
        Bnew(n,m)=transpose(U(:,n))*conj(U(:,m));
        Anew(n,m)=transpose(U(:,n))*A*conj(U(:,m));
    end
end

%Eigenvalue computations
[VI,E]=eig(mid(Anew),mid(Bnew));
Evec=diag(E);
Evec(1:5)
[ESI,I]=sort(Evec);

L=intval(zeros(M,1));
for k=1:M
    v=VI(:,I(k));
    [L(k),X]=verifyeig(Anew,ESI(k),v,Bnew);
end
```