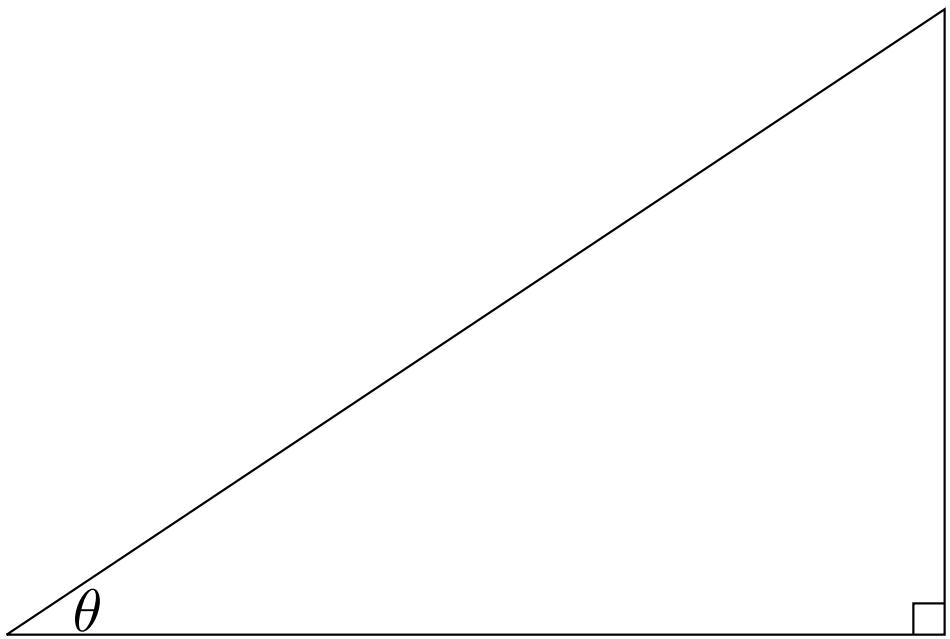
Audio-Visual Computing

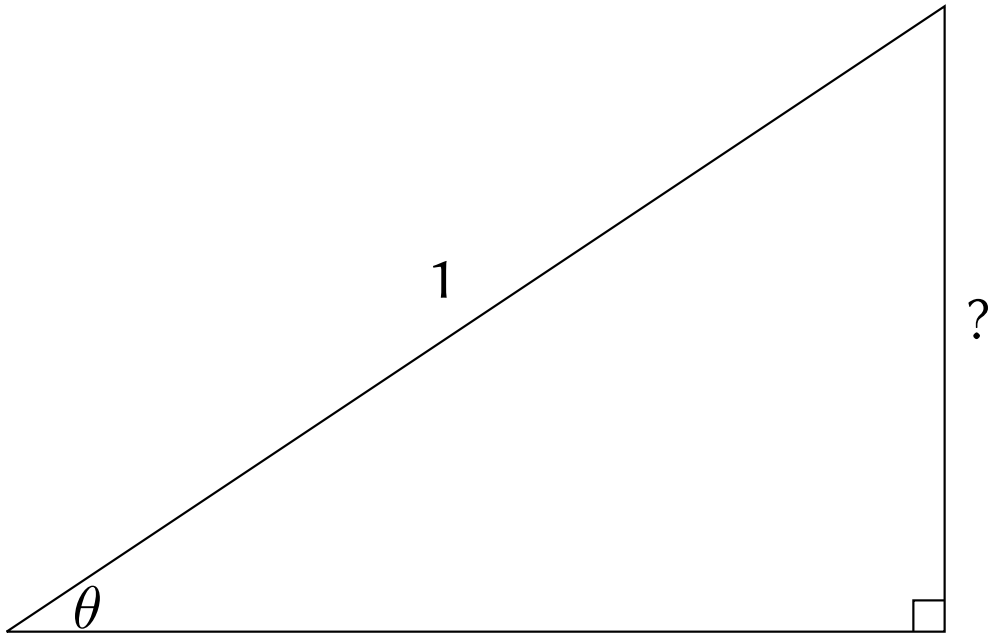
Christophe Rhodes
c.rhodes@gold.ac.uk

Winter 2015

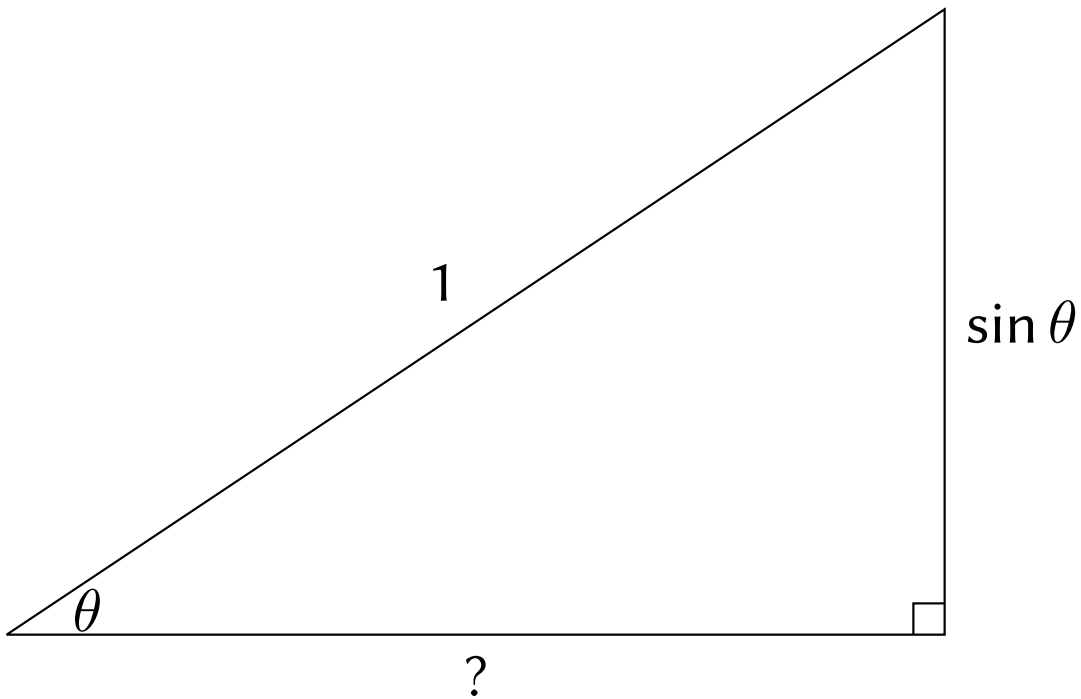
Revision: trigonometry



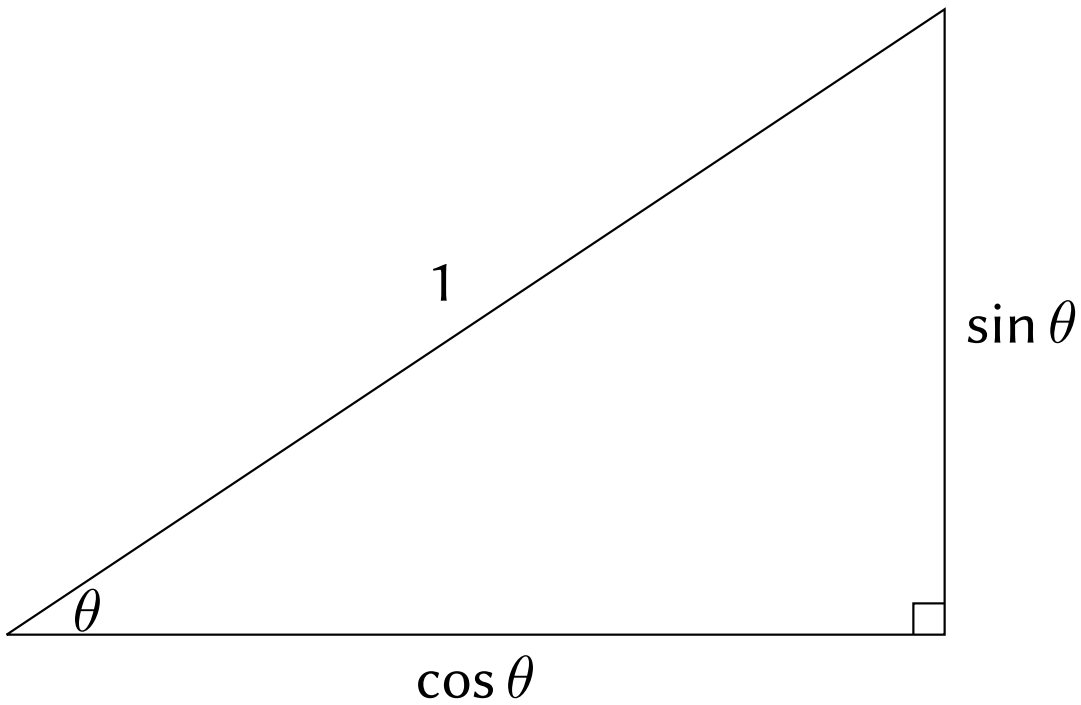
Revision: trigonometry



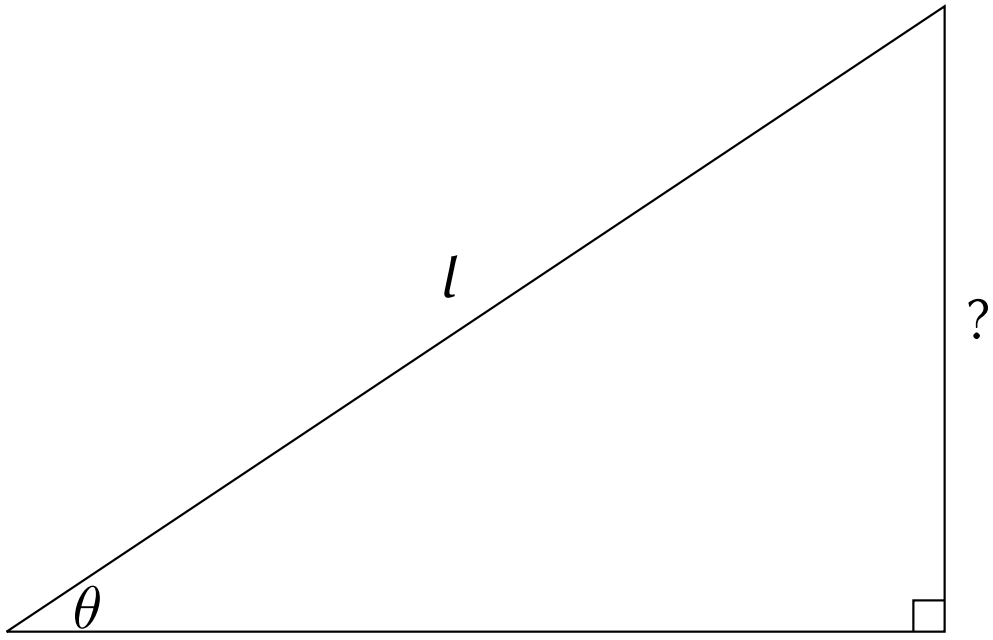
Revision: trigonometry



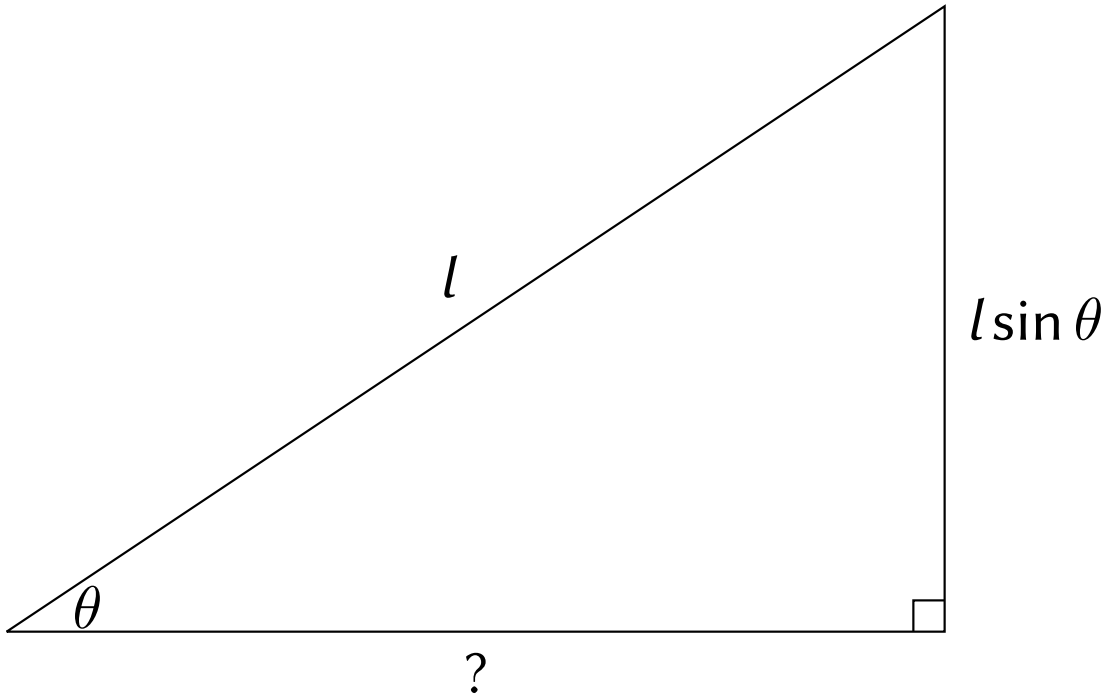
Revision: trigonometry



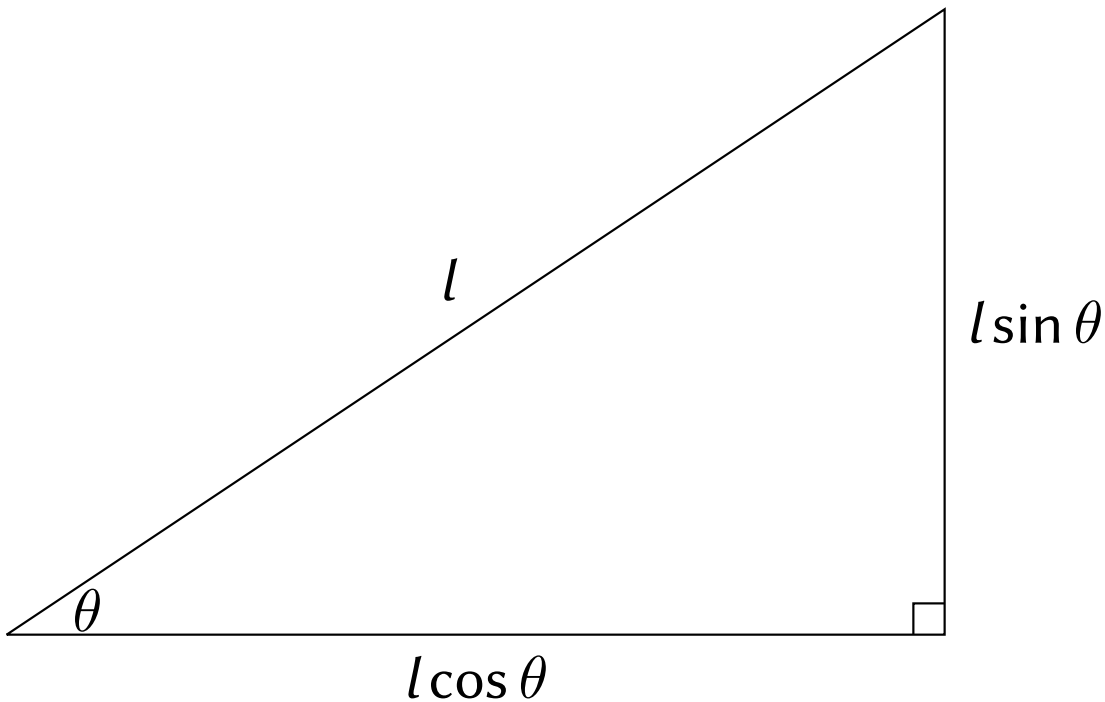
Revision: trigonometry



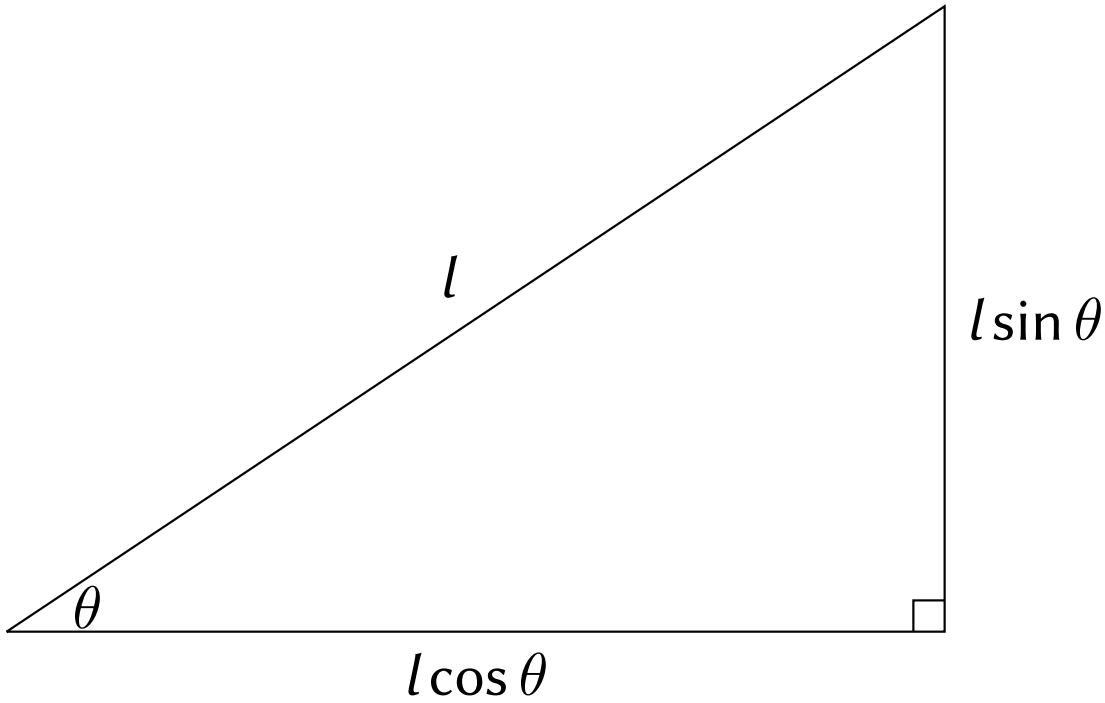
Revision: trigonometry



Revision: trigonometry

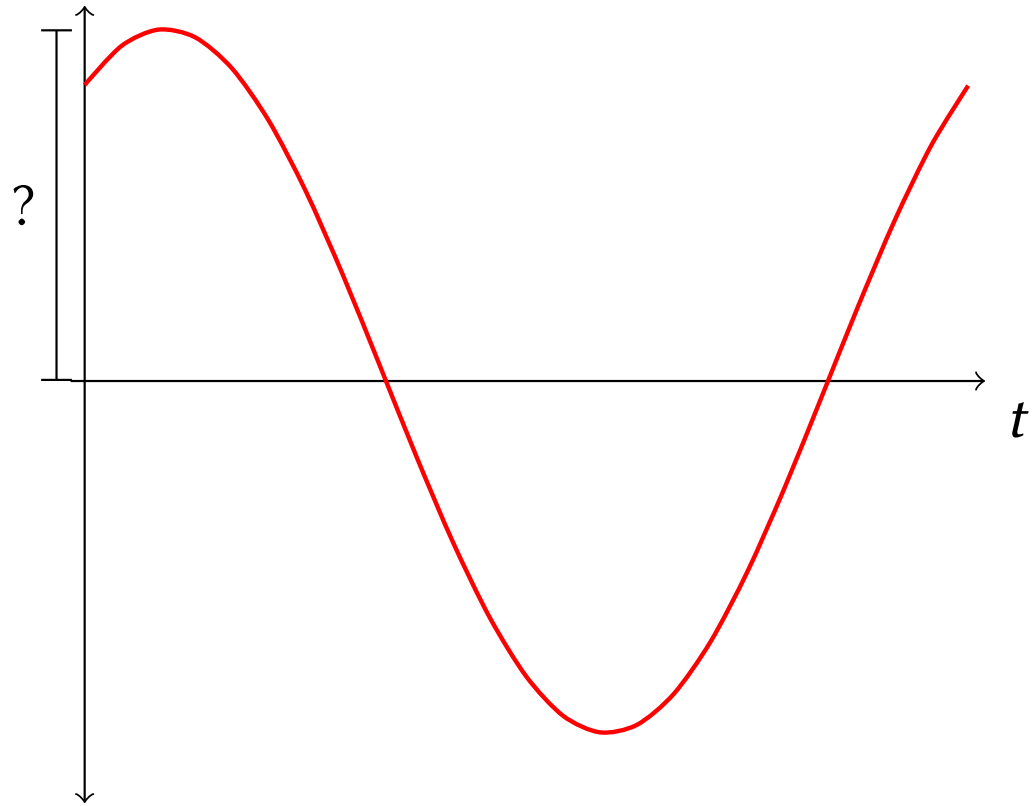


Revision: trigonometry



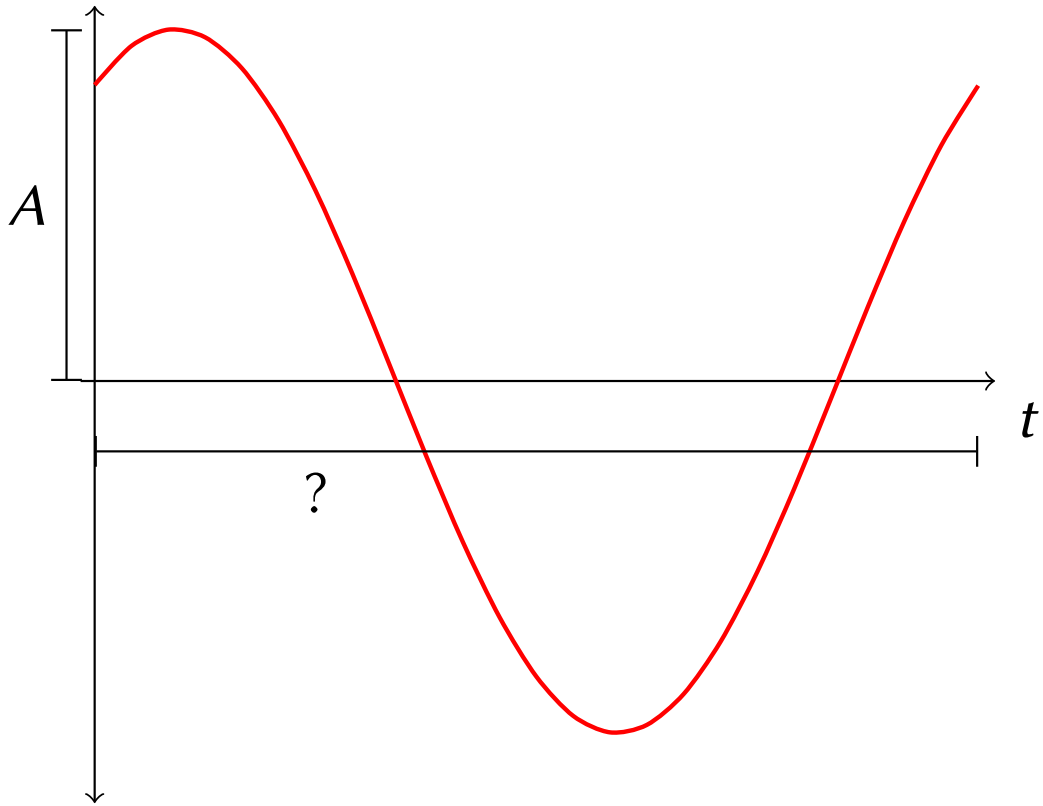
$$\sin = \frac{\text{opp}}{\text{hyp}}; \cos = \frac{\text{adj}}{\text{hyp}}$$

Revision: sinusoidal oscillation



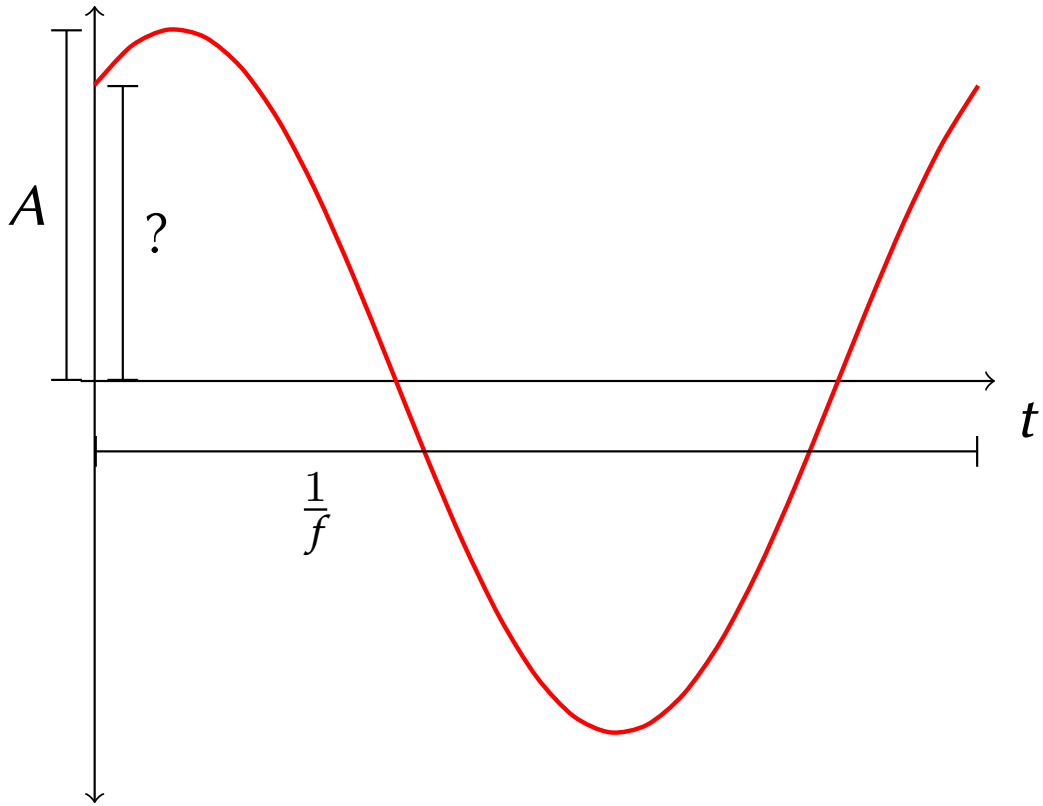
$$y = A\sin(2\pi ft + \phi)$$

Revision: sinusoidal oscillation



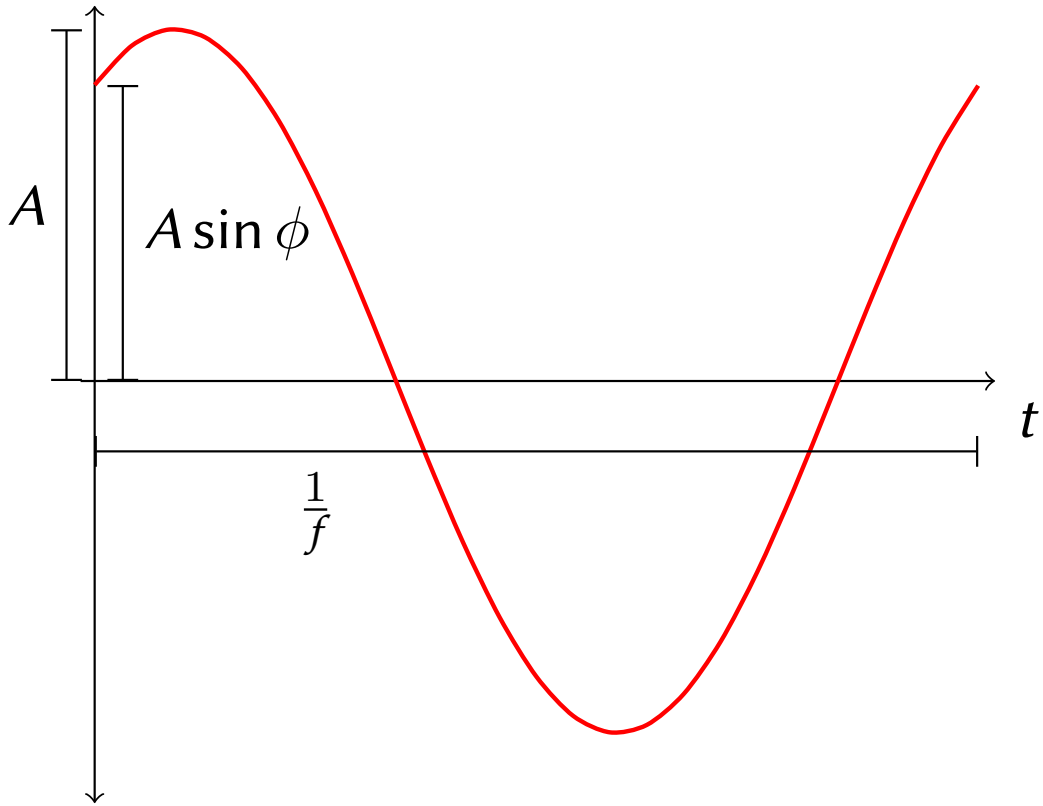
$$y = A\sin(2\pi ft + \phi)$$

Revision: sinusoidal oscillation



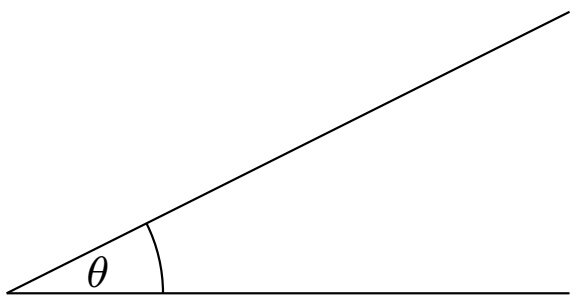
$$y = A\sin(2\pi ft + \phi)$$

Revision: sinusoidal oscillation

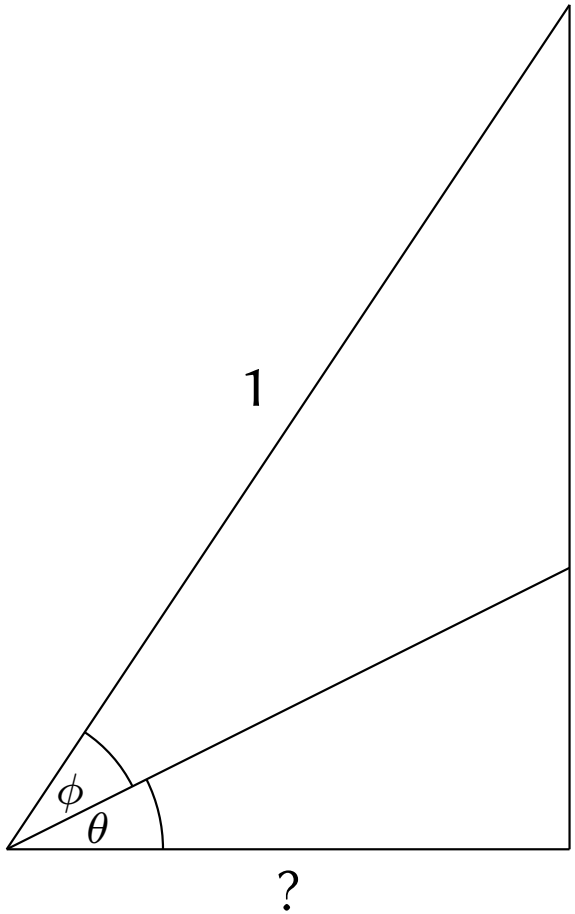


$$y = A\sin(2\pi ft + \phi)$$

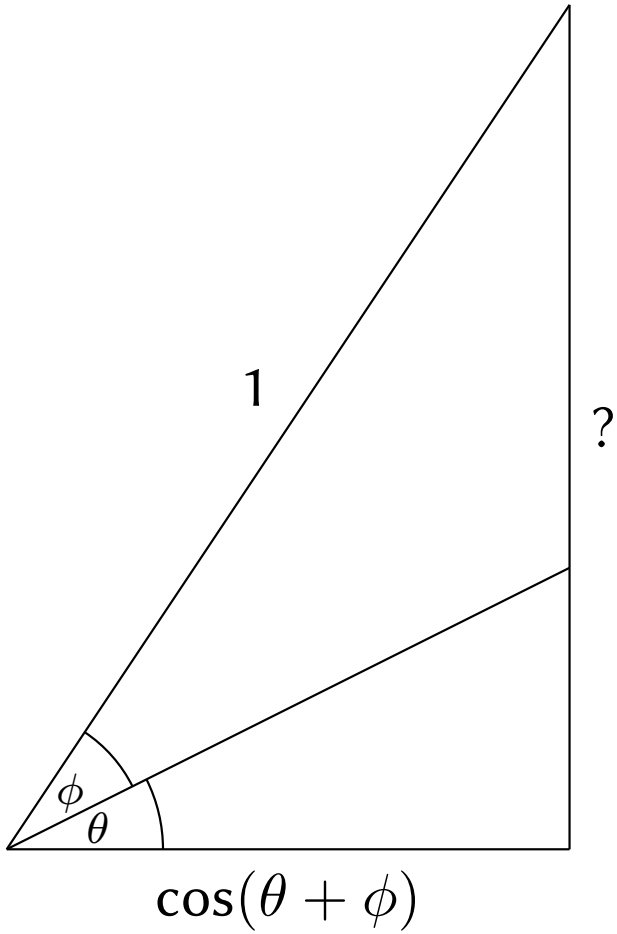
$\theta + \phi$



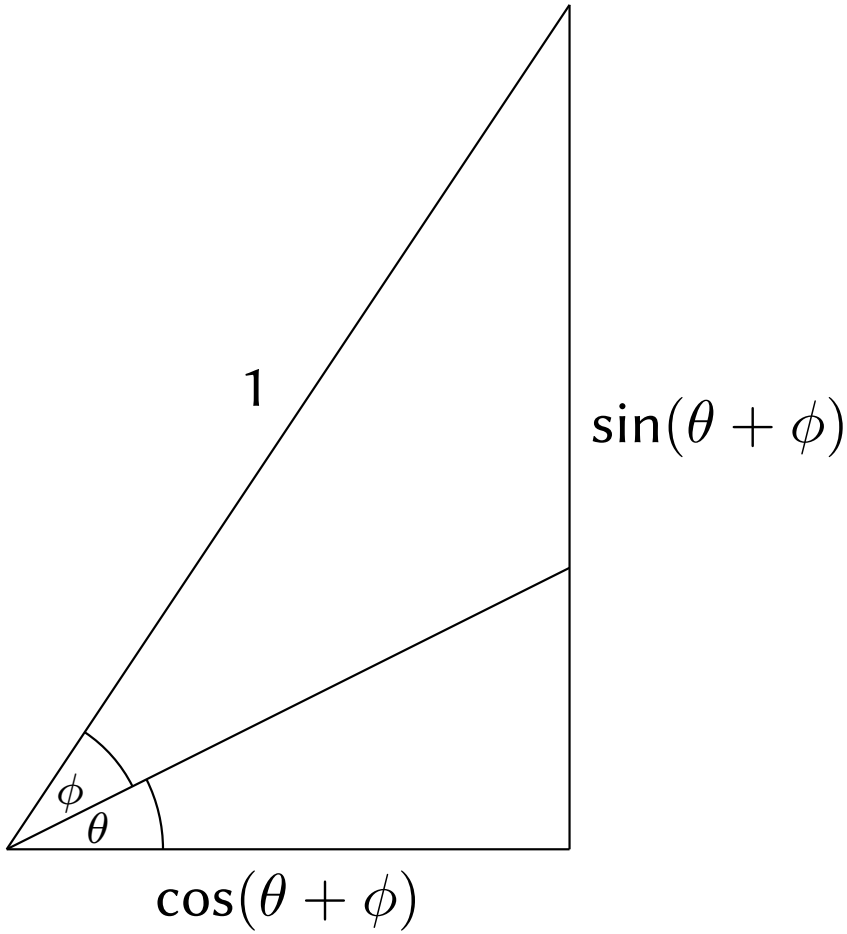
$\theta + \phi$



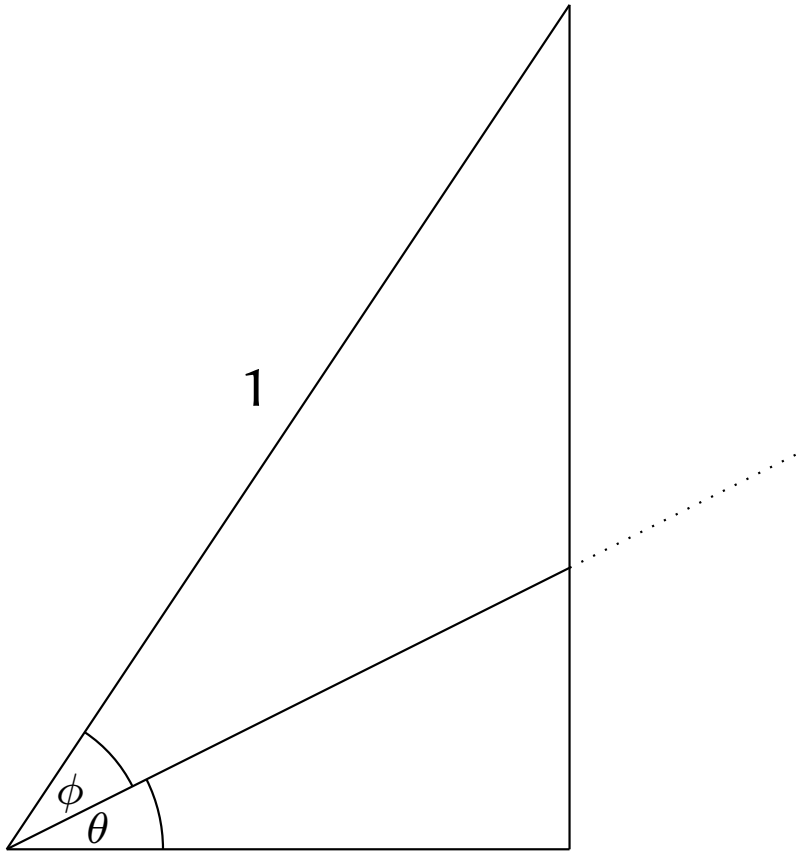
$\theta + \phi$



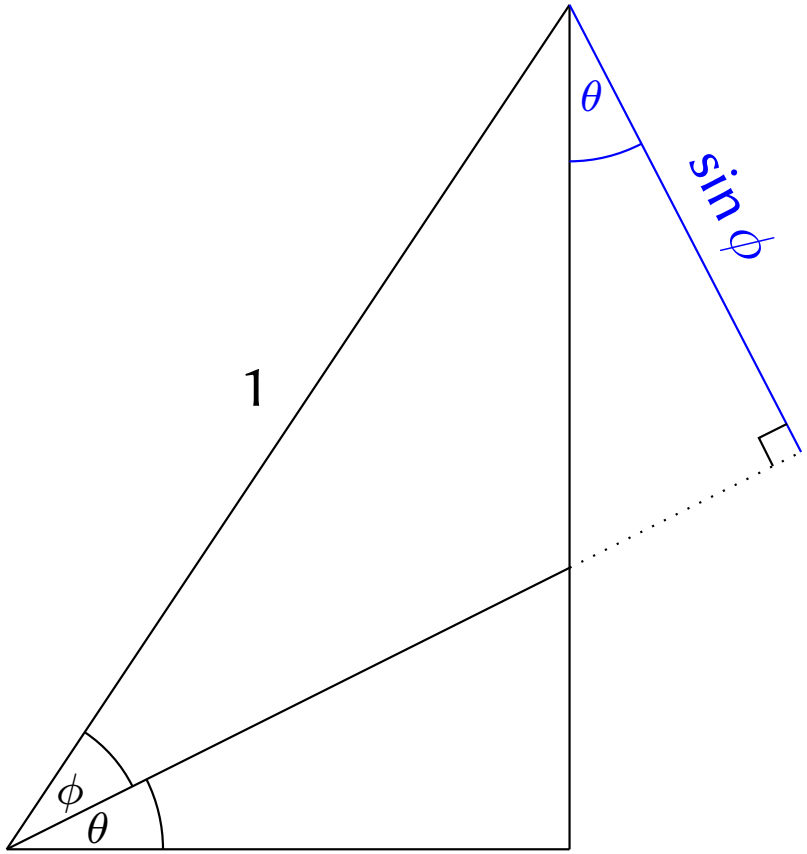
$\theta + \phi$



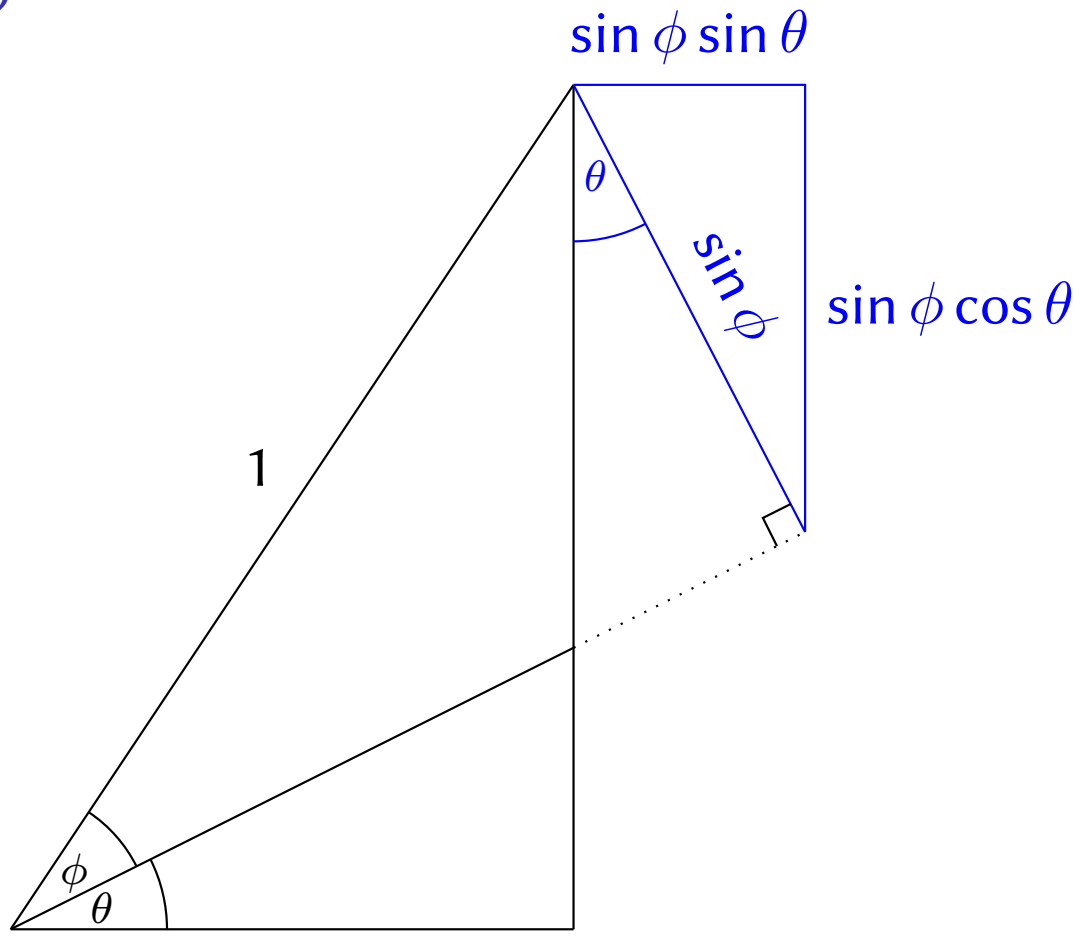
$\theta + \phi$



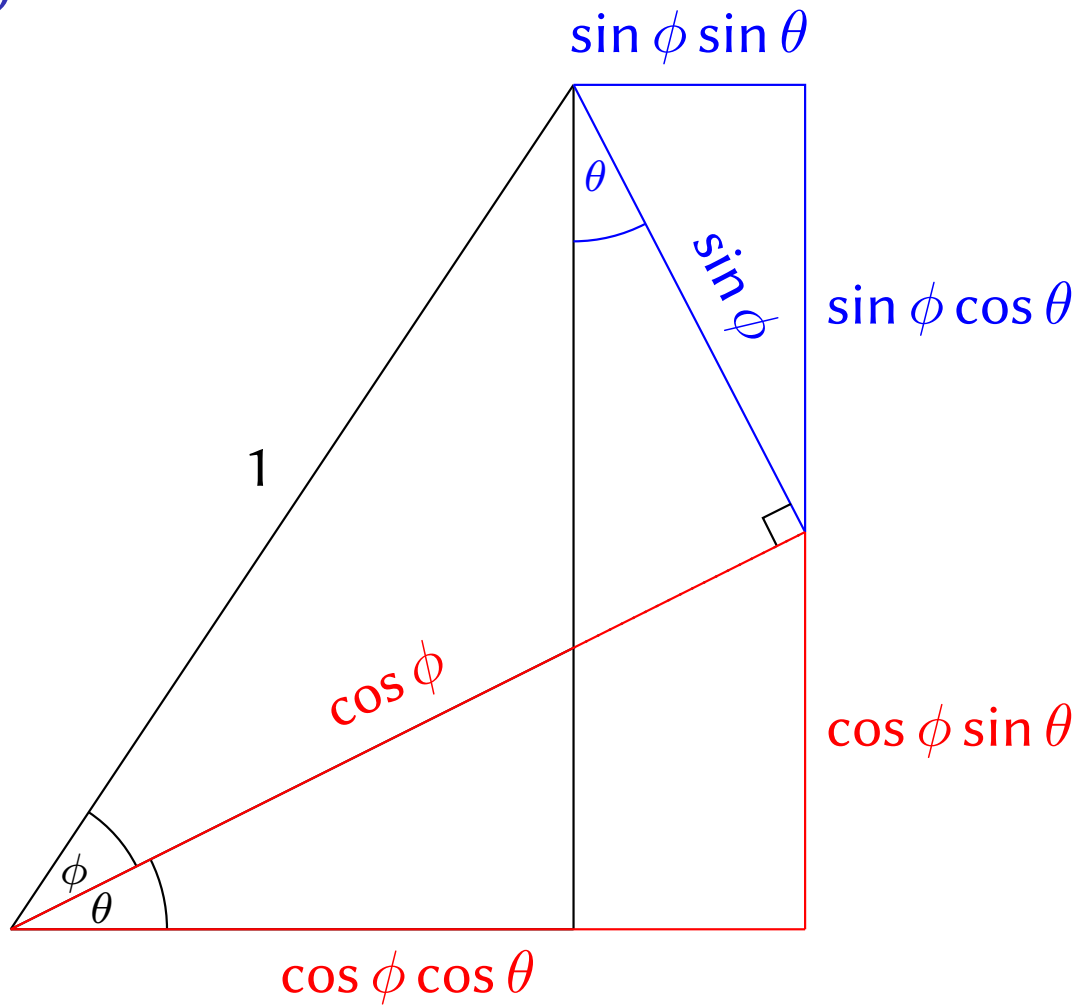
$\theta + \phi$



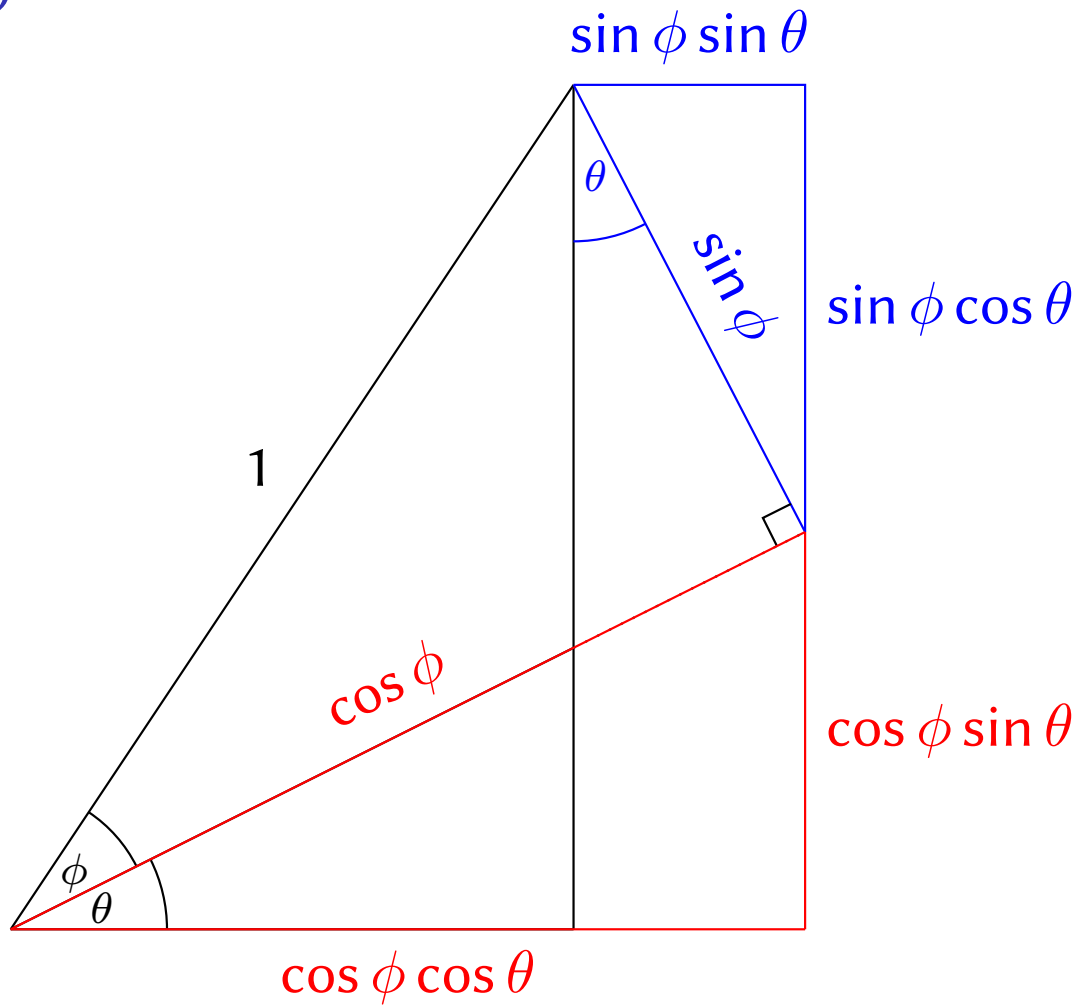
$\theta + \phi$



$\theta + \phi$



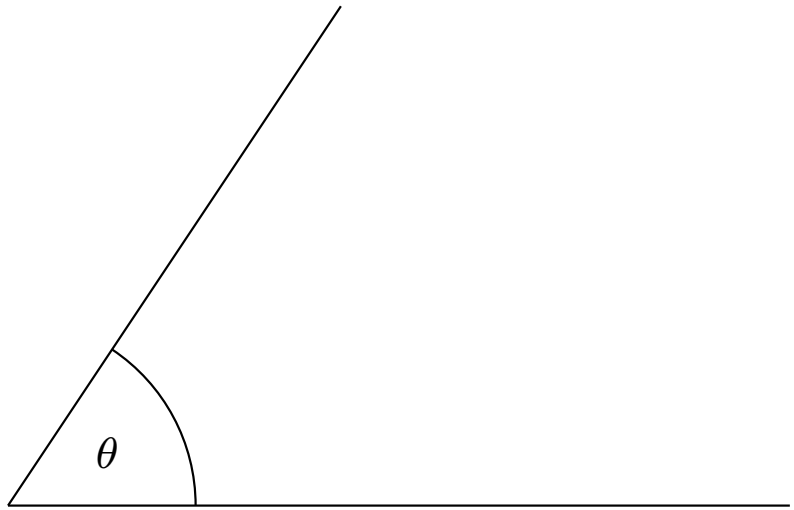
$\theta + \phi$



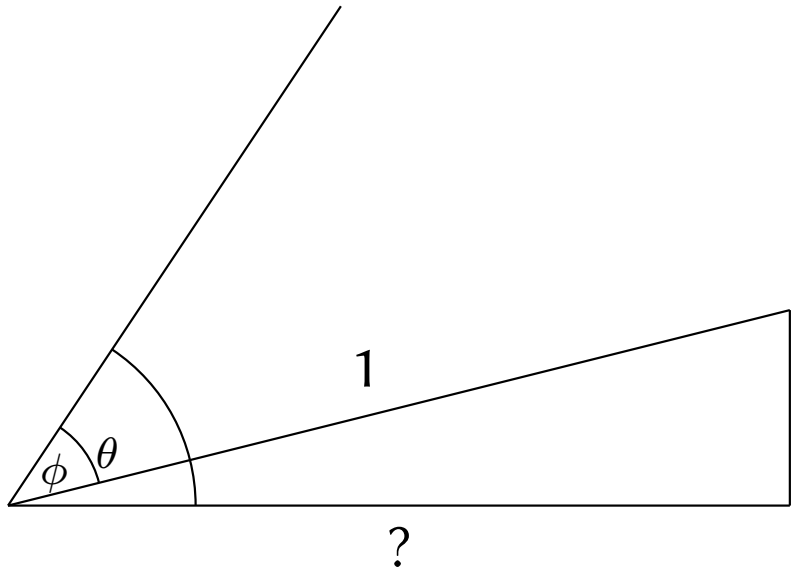
$$\sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$

$$\cos(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi$$

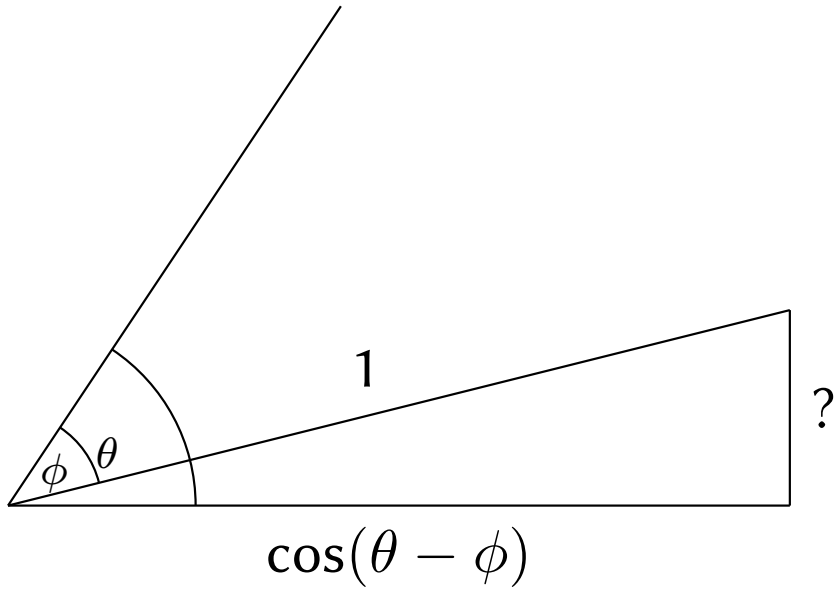
$\theta - \phi$



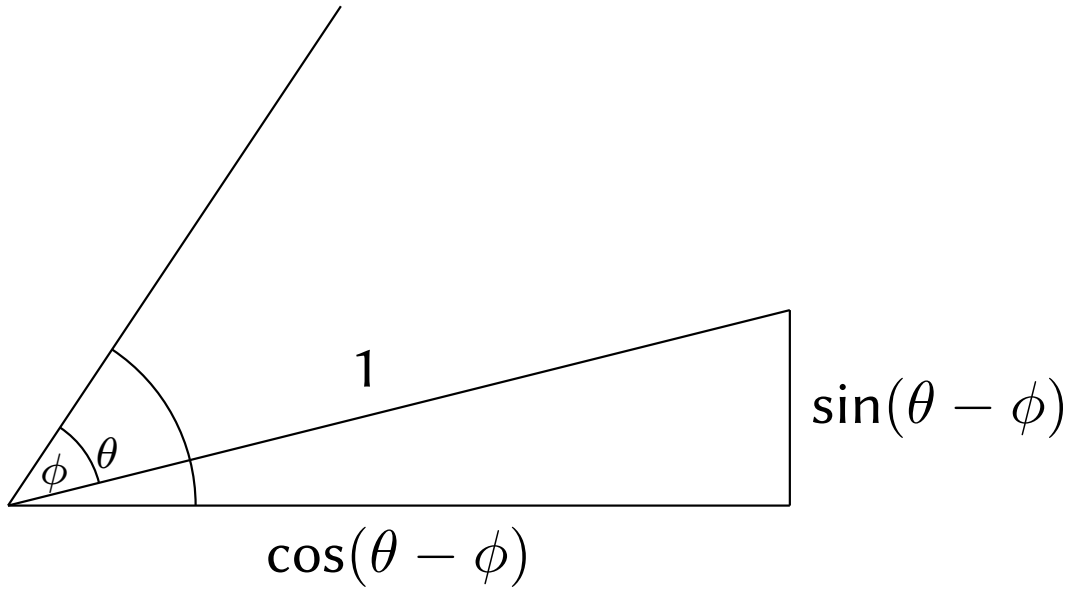
$\theta - \phi$



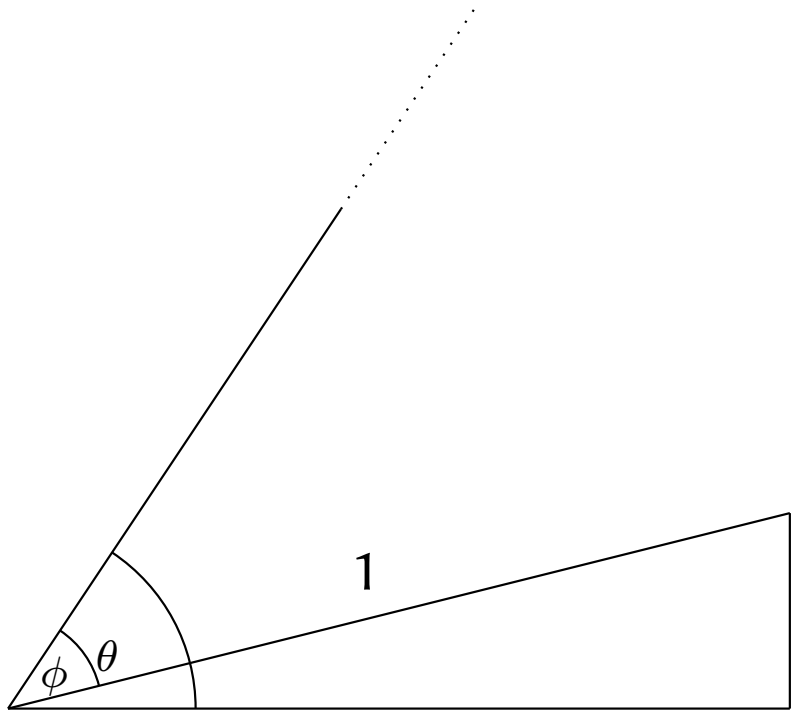
$\theta - \phi$



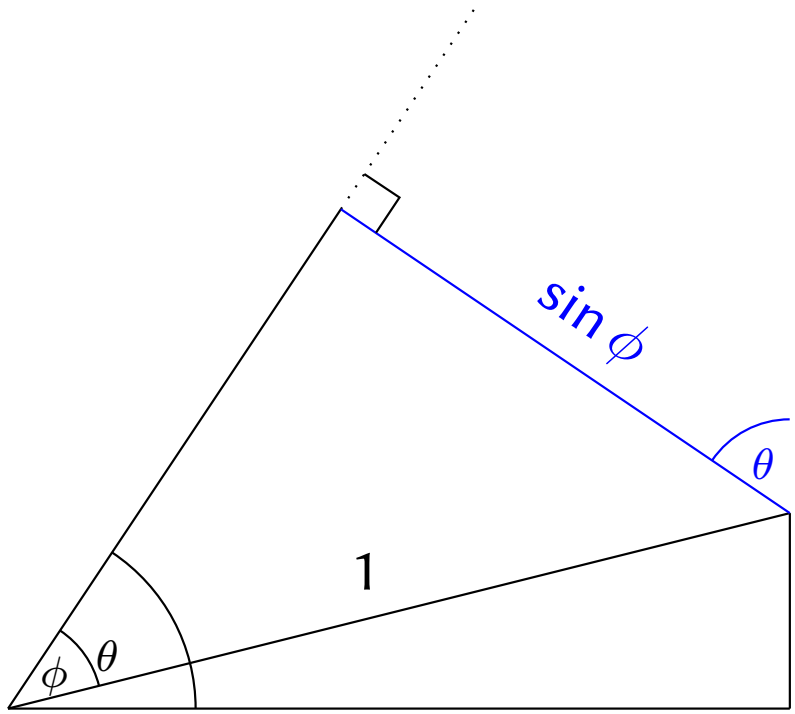
$\theta - \phi$



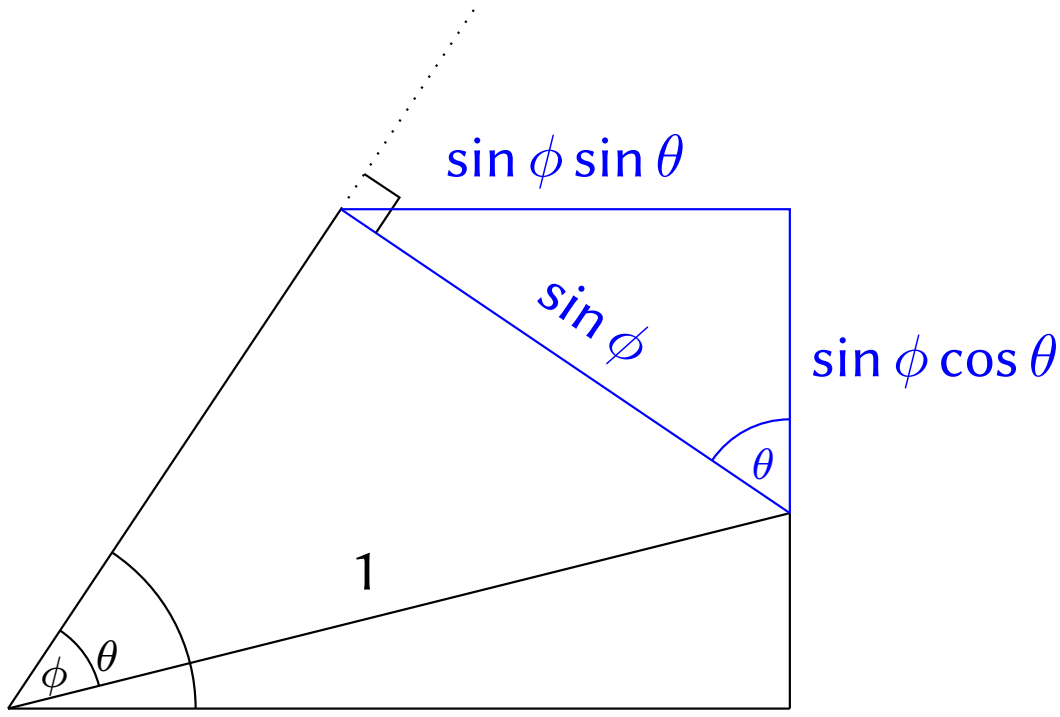
$\theta - \phi$



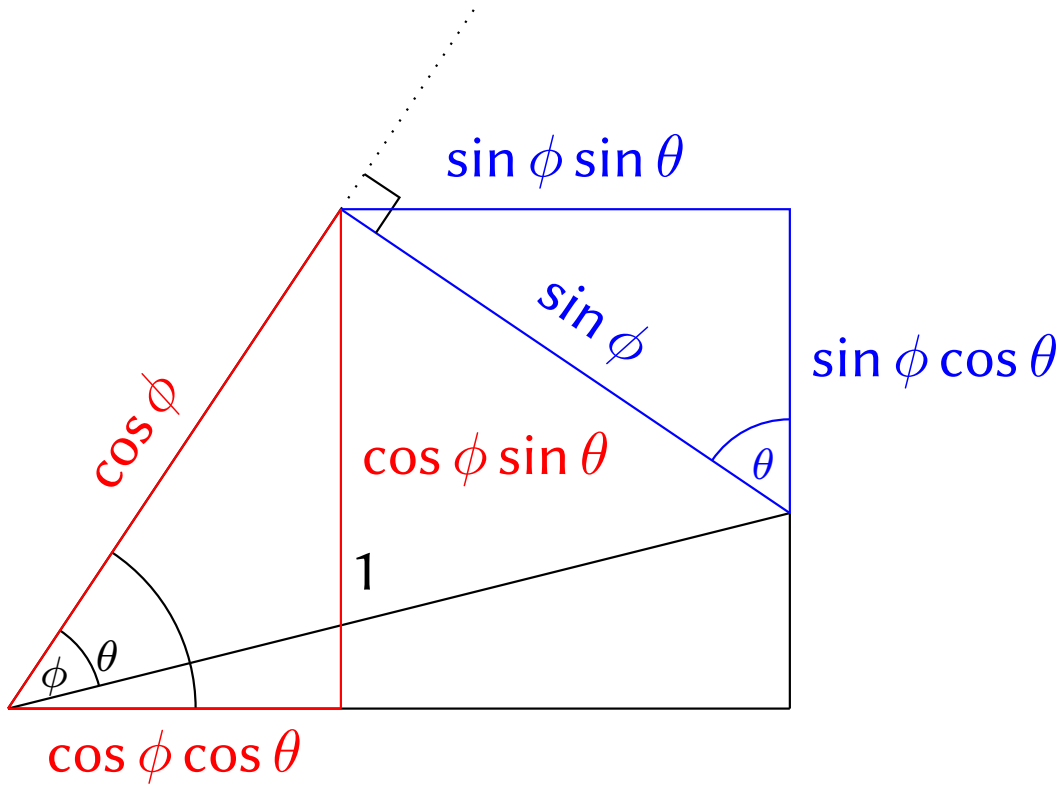
$\theta - \phi$



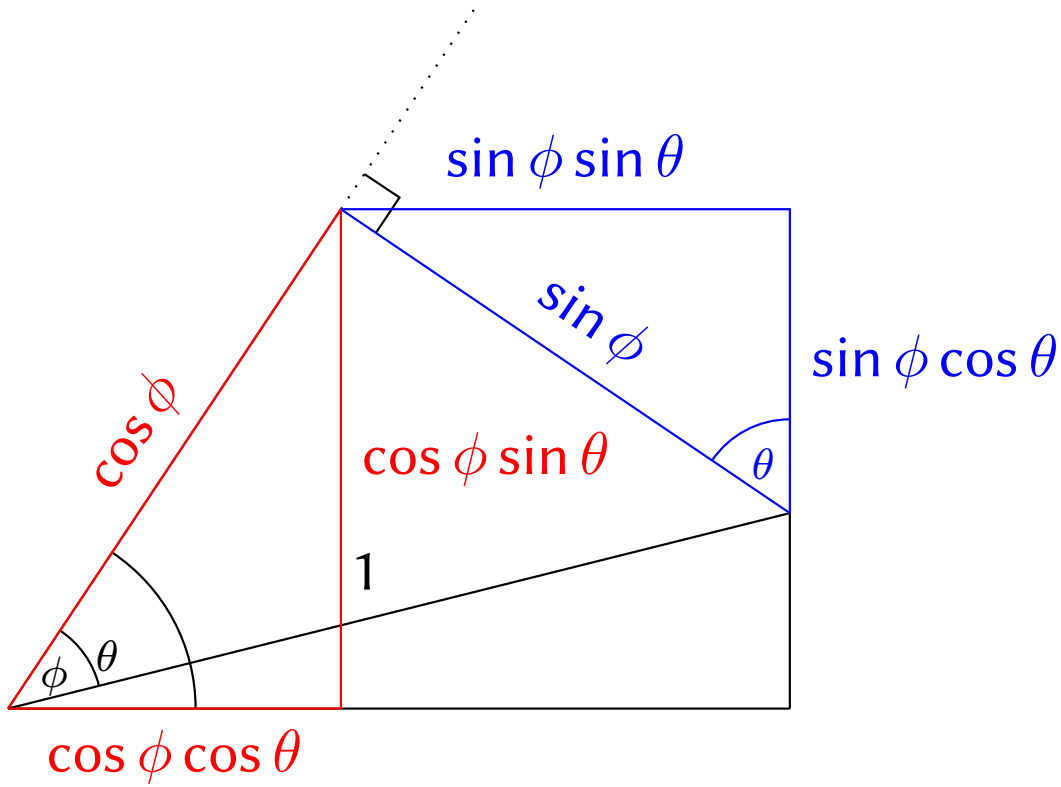
$\theta - \phi$



$\theta - \phi$



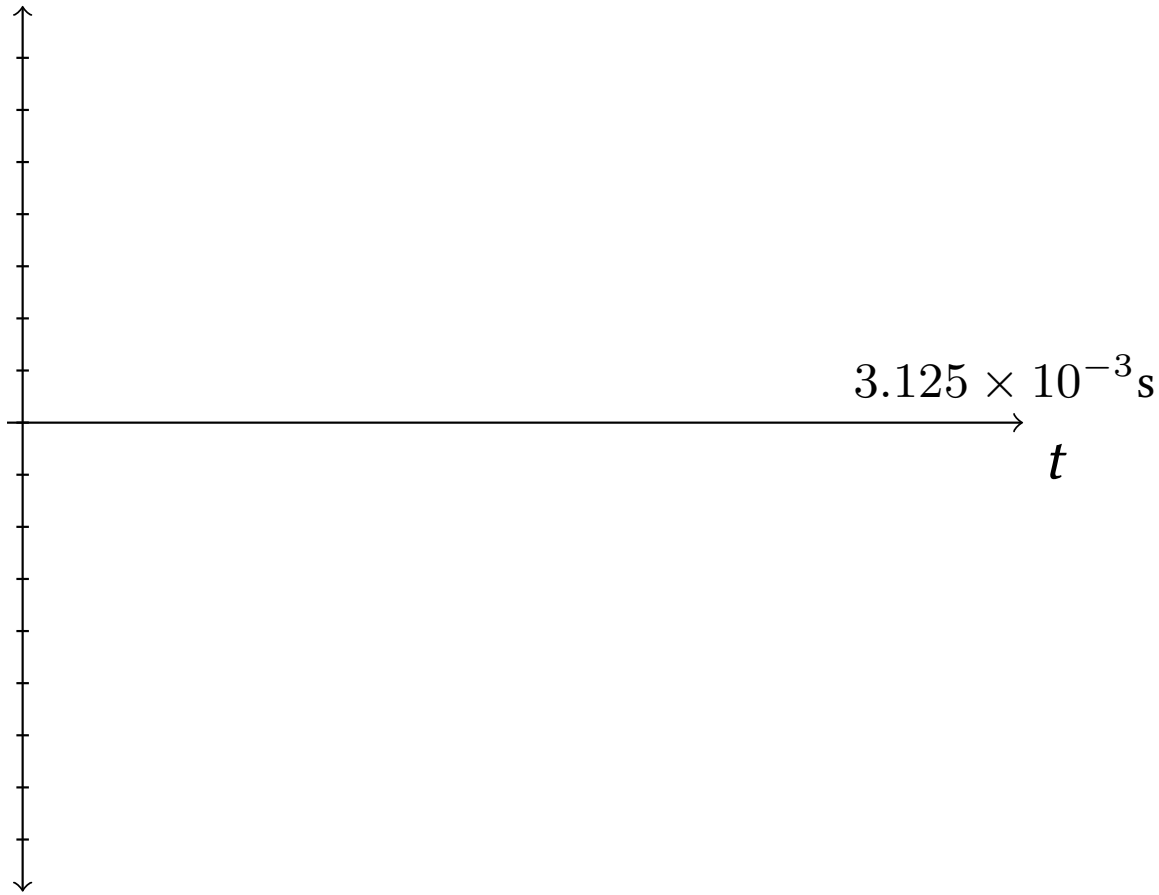
$\theta - \phi$



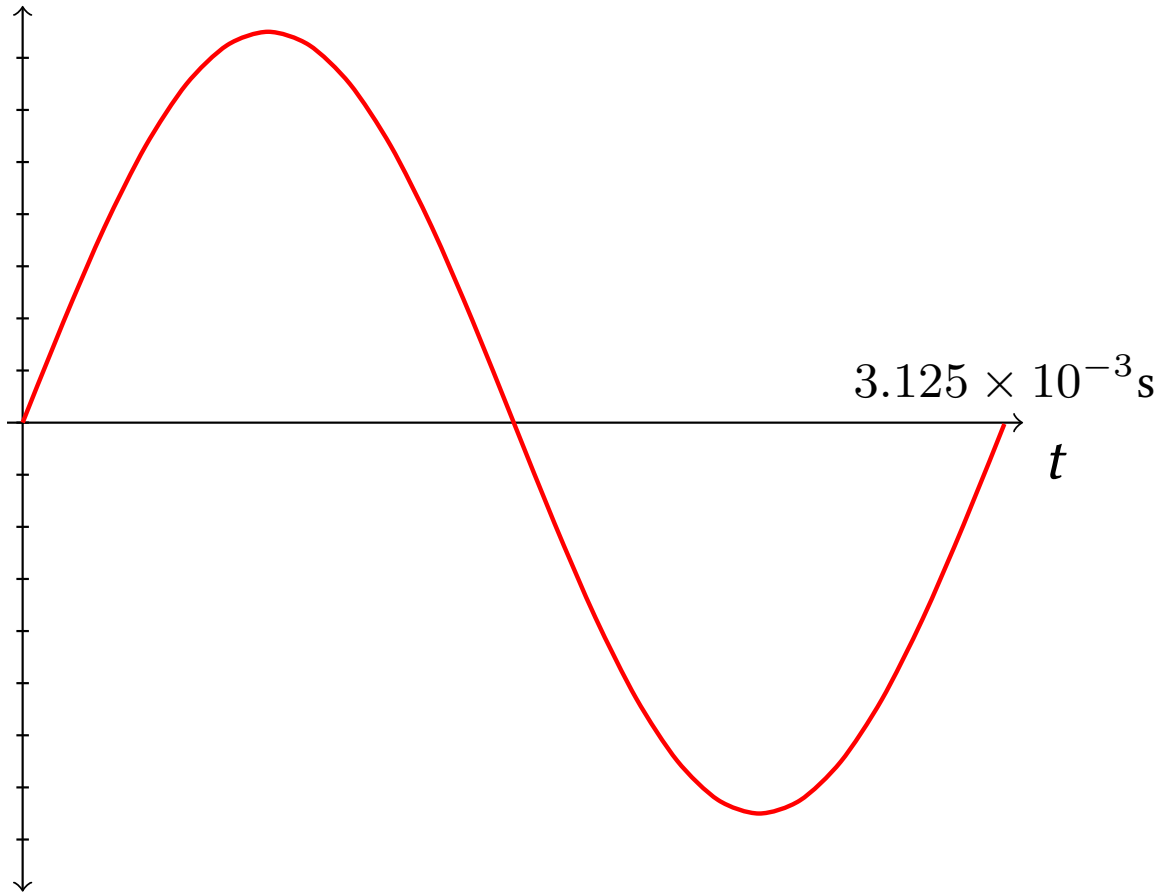
$$\sin(\theta - \phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$$

$$\cos(\theta - \phi) = \cos\theta\cos\phi + \sin\theta\sin\phi$$

Sampling and bit-depth

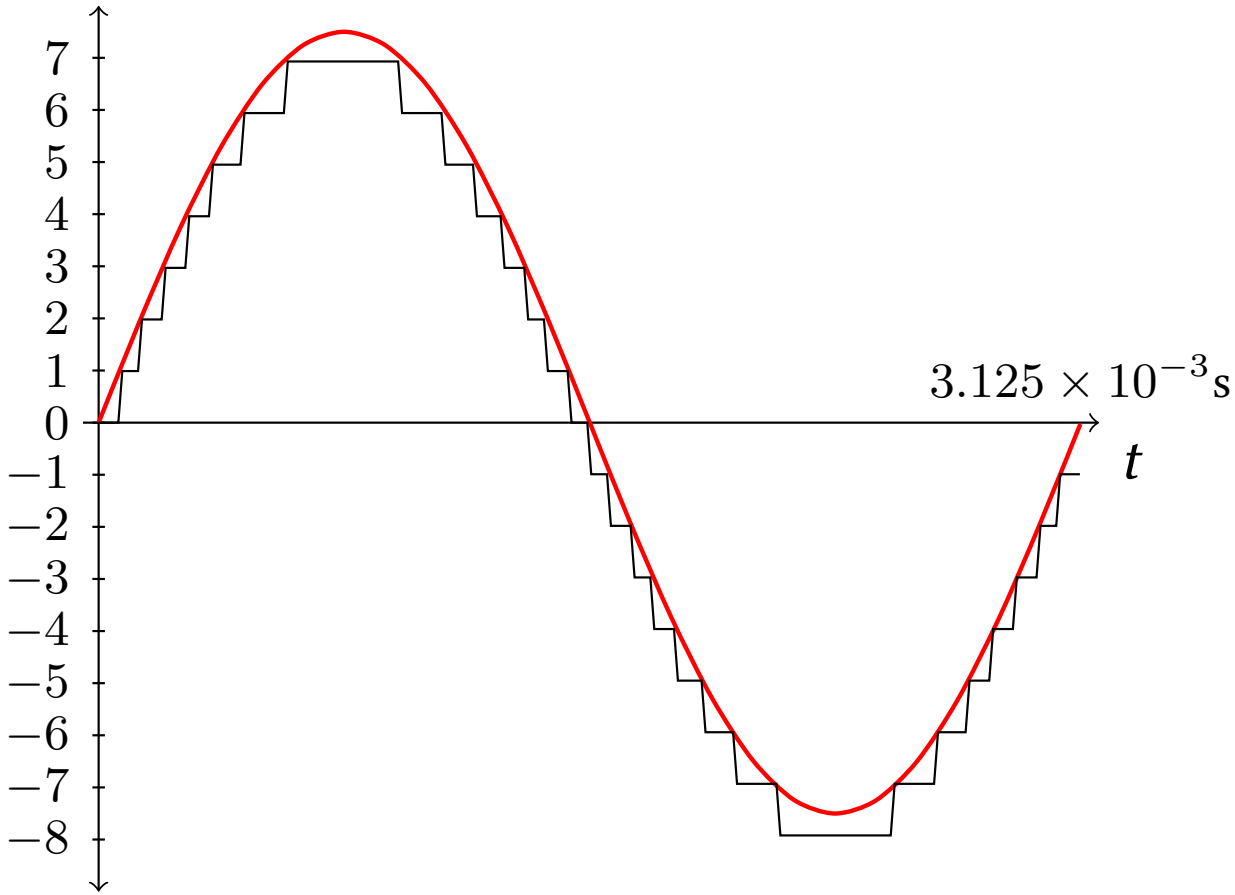


Sampling and bit-depth



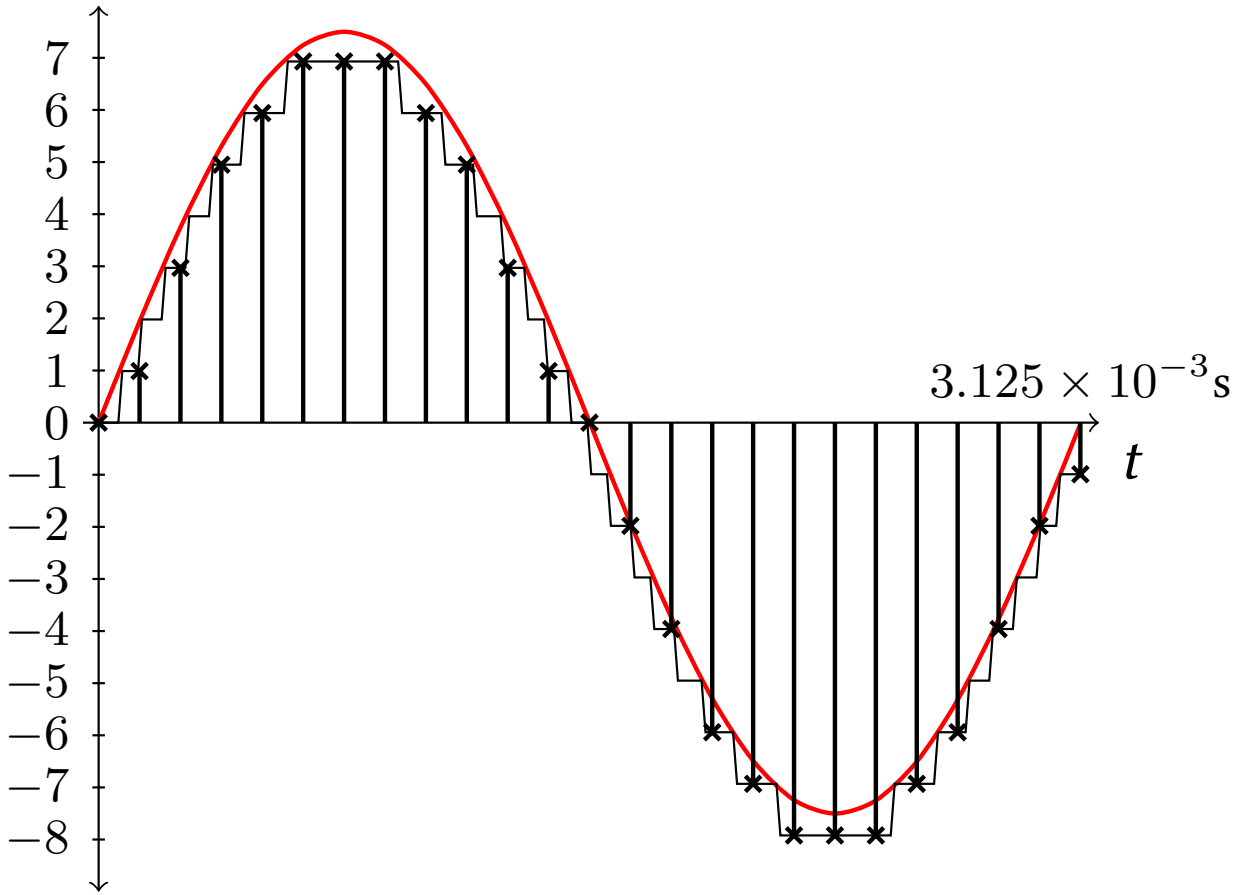
- wave frequency:

Sampling and bit-depth



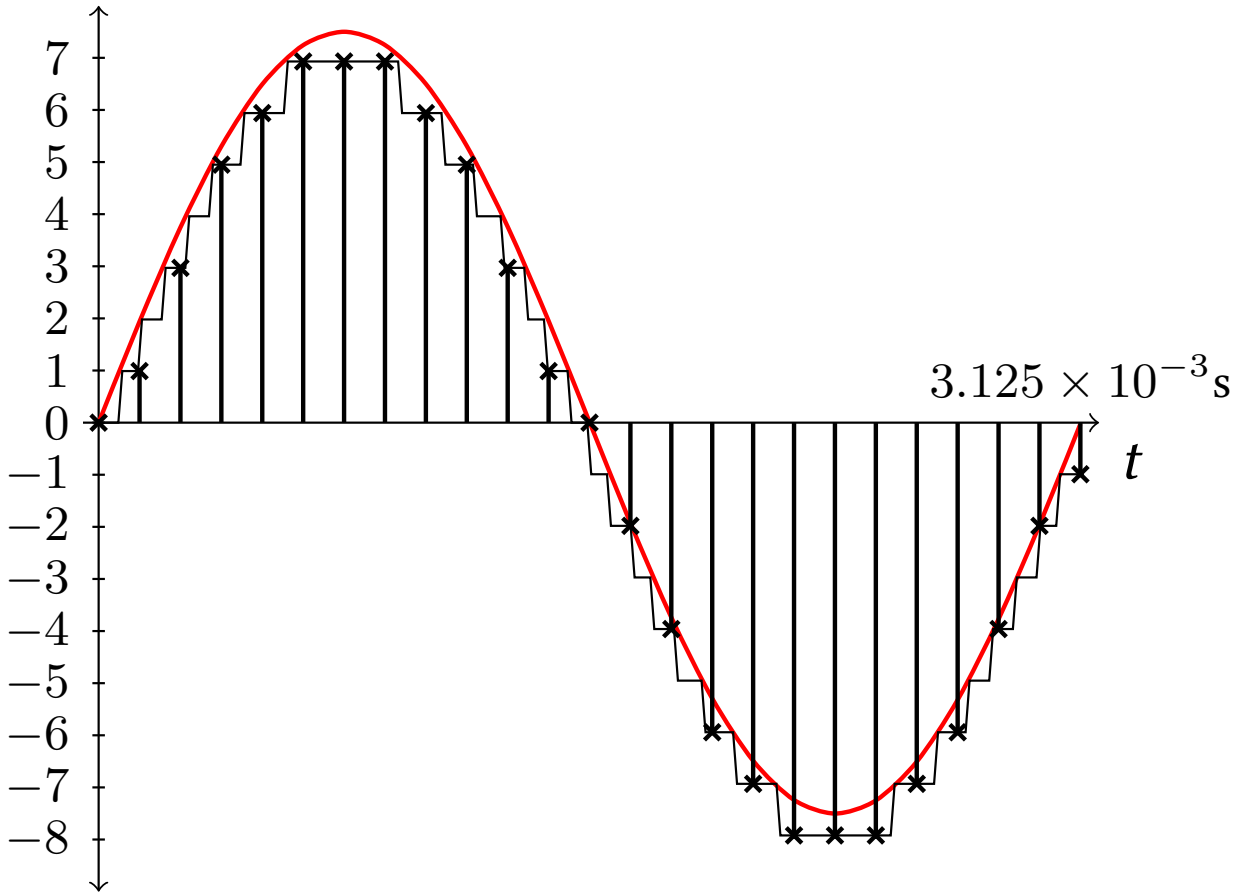
- wave frequency: 320Hz
- bit depth:

Sampling and bit-depth



- wave frequency: 320Hz
- bit depth: 4 bits
- sample rate:

Sampling and bit-depth



- wave frequency: 320Hz
- bit depth: 4 bits
- sample rate: 8000Hz

Synthesis in minim

- `import ddf.minim.*;`
- interface `AudioSignal`
  - constructor
  - `void generate(float [] buf)`
  - `void generate(float [] left, float [] right)`

Sinusoidal synthesis

Driver (main file):

```
import ddf.minim.*;

Minim minim;
Sine sine;

void setup() {
  minim = new Minim(this);
  sine = new Sine(440);
  AudioOutput out = minim.getLineOut(Minim.STEREO, 4096);
  out.addSignal(sine);
}

void stop() {
  minim.stop();
  super.stop();
}

void draw() {
}
```

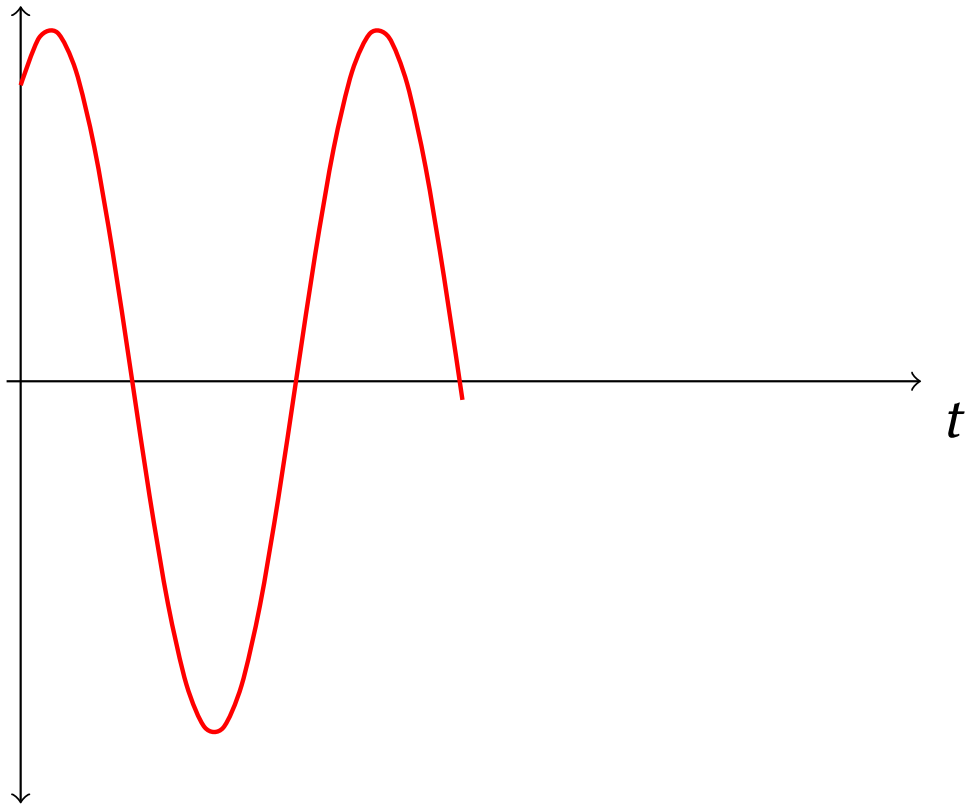
Sinusoidal synthesis

Signal implementation:

```
class Sine1 implements AudioSignal {
  float f;
  Sine1(float frequency) { f = frequency; }
  void generate(float [] buf) {
    for (int i = 0; i < buf.length; i++)
      buf[i] = sin(2*PI*f*i / 44100);
  }
  void generate(float [] left, float [] right) {
    for (int i = 0; i < left.length; i++) {
      left[i] = sin(2*PI*f*i / 44100);
      right[i] = left[i];
    }
  }
}
```

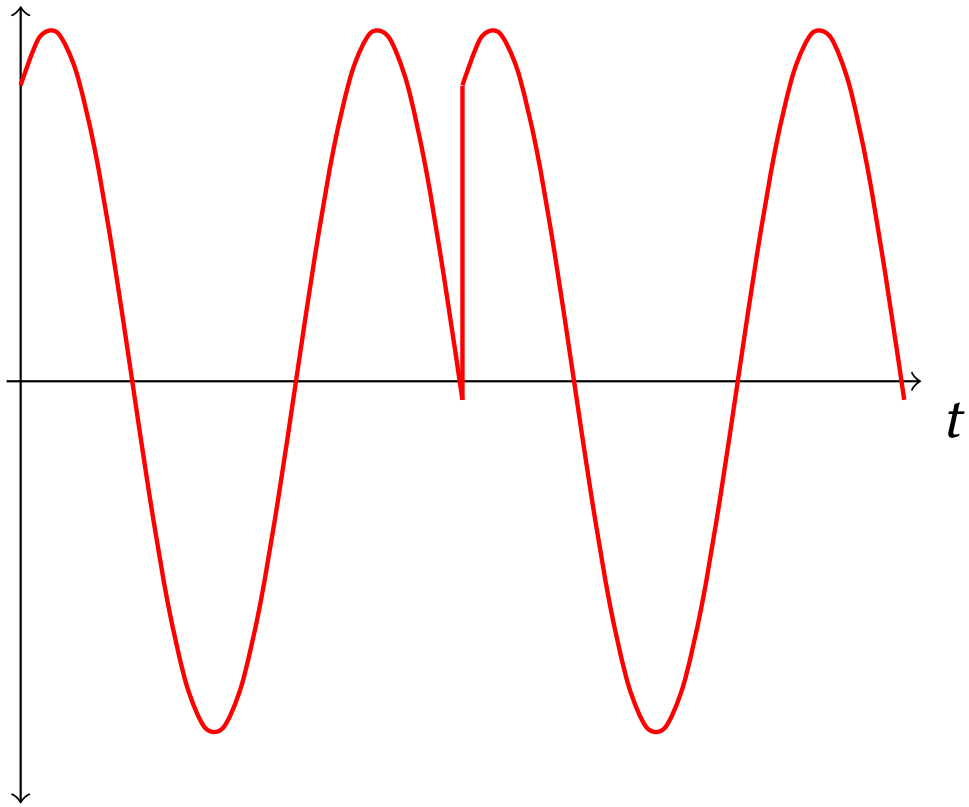
Sinusoidal synthesis

Continuity of phase



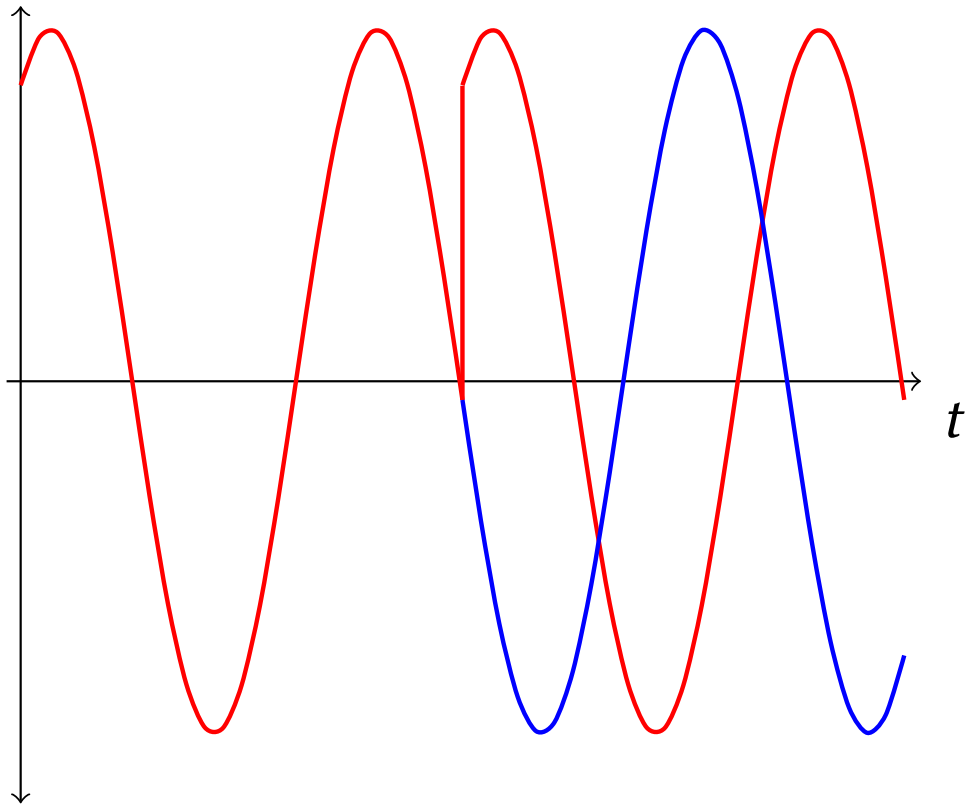
Sinusoidal synthesis

Continuity of phase



Sinusoidal synthesis

Continuity of phase



- phase at end of buffer saved to offset phase at start of next buffer.

Sinusoidal synthesis

Signal implementation (take 2):

```
class Sine implements AudioSignal {
  float f;
  float p = 0;
  Sine(float frequency) { f = frequency; }
  void generate(float [] buf) {
    for (int i = 0; i < buf.length; i++)
      buf[i] = sin(2*PI*f*i / 44100 + p);
    p += 2*PI*f*buf.length/44100;
    p = p - 2*PI*floor(p/(2*PI));
  }
  void generate(float [] left, float [] right) {
    for (int i = 0; i < left.length; i++) {
      left[i] = sin(2*PI*f*i / 44100 + p);
      right[i] = left[i];
    }
    p += 2*PI*f*left.length/44100;
    p = p - 2*PI*floor(p/(2*PI));
  }
}
```

Sinusoidal synthesis

Variable frequency

Signal implementation (take 3):

```
class Sine implements AudioSignal {
  float f;
  float p = 0;
  Sine(float frequency) { f = frequency; }
  void generate(float [] buf) {
    for (int i = 0; i < buf.length; i++)
      buf[i] = sin(2*PI*f*i / 44100 + p);
    p += 2*PI*f*buf.length/44100;
    p = p - 2*PI*floor(p/(2*PI));
  }
  void generate(float [] left, float [] right) {
    for (int i = 0; i < left.length; i++) {
      left[i] = sin(2*PI*f*i/44100 + p);
      right[i] = left[i];
    }
    p += 2*PI*f*left.length/44100;
    p = p - 2*PI*floor(p/(2*PI));
  }
  float getf() { return f; }
  void setf(float fnew) { f = fnew; }
}
```

Sinusoidal synthesis

Variable frequency

Driver code:

```
import ddf.minim.*;

Minim minim;
Sine sine;

void setup() {
  minim = new Minim(this);
  sine = new Sine(440);
  AudioOutput out = minim.getLineOut(Minim.STEREO, 512);
  out.addSignal(sine);
}

void stop() {
  minim.stop(); super.stop();
}

void draw() { }

void keyPressed() {
  if(key == CODED) {
    switch(keyCode) {
    case RIGHT: case UP:
      sine.setf(sine.getf()*pow(2,1.0/12)); break;
    case LEFT: case DOWN:
      sine.setf(sine.getf()/pow(2,1.0/12)); break;
    default:
    }
  }
}
```

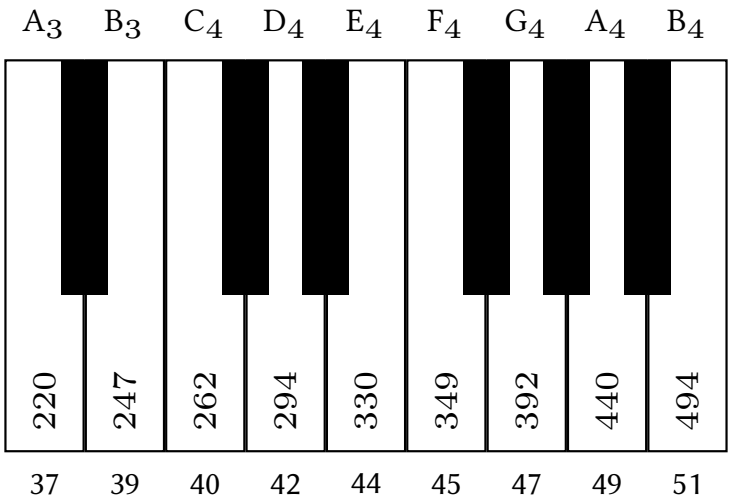
Sinusoidal synthesis

Frequency sensitivity

Notes:

- buffer size and latency
- frequency sensitivity dependence

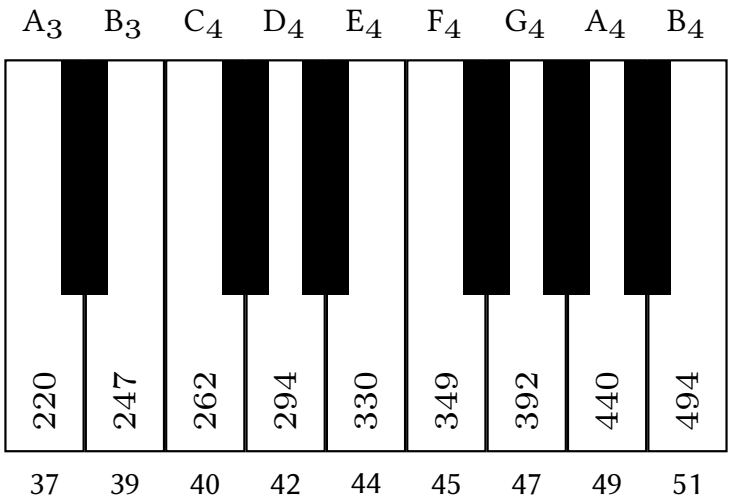
12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling

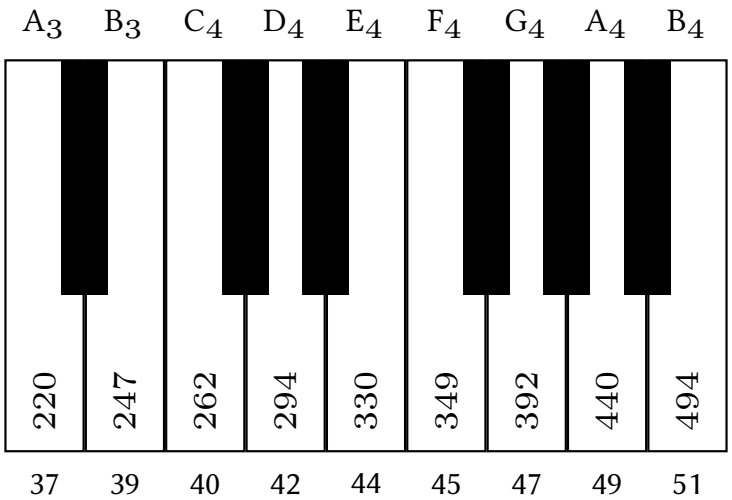
12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling
- what’s “one twelfth of a doubling”?

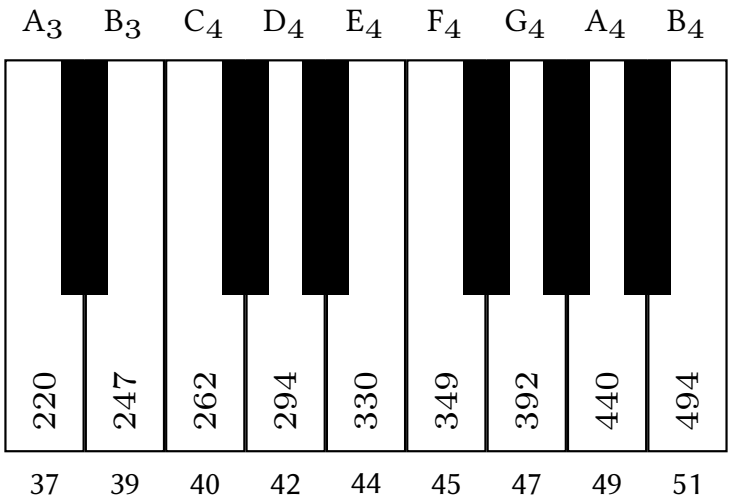
12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling
- what’s “one twelfth of a doubling”?
  - $f(x) = \alpha \times x$

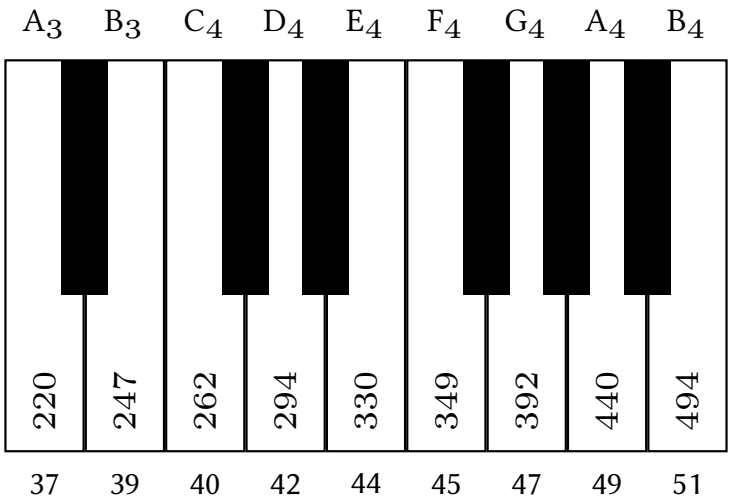
12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling
- what’s “one twelfth of a doubling”?
  - $f(x) = \alpha \times x$
  - $f(f(x)) = \alpha^2 \times x$

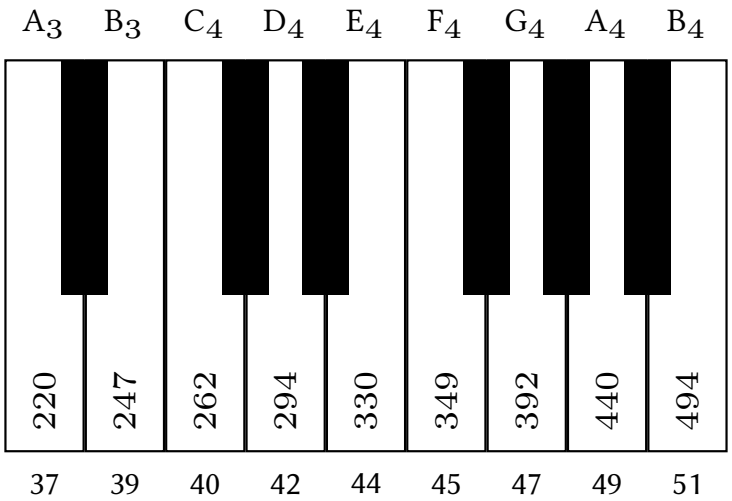
12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling
- what’s “one twelfth of a doubling”?
  - $f(x) = \alpha \times x$
  - $f(f(x)) = \alpha^2 \times x$
  - $f(...(f(x))) = \alpha^{12} \times x$

12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
  - one octave: frequency doubling;
  - one semitone: one twelfth of a doubling
- what’s “one twelfth of a doubling”?
  - $f(x) = \alpha \times x$
  - $f(f(x)) = \alpha^2 \times x$
  - $f(...(f(x))) = \alpha^{12} \times x$
  - $\alpha = 2^{\frac{1}{12}}$

```
import ddf.minim.*;

Minim minim;
Sine sine;
int [] tones = { 14, 12, 14, 9, 5, 9, 2, 2 };

void setup() {
  minim = new Minim(this);
  sine = new Sine(440);
  AudioOutput out = minim.getLineOut(Minim.MONO, 512);
  out.addSignal(sine);
}

void stop() {
  minim.stop();
  super.stop();
}

void draw() {
  sine.setf(440 * pow(2,tones[floor(millis()/500) % tones.length]/12.0));
}
```

Additive Synthesis

Beats

$$\sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$

$$\sin(\theta - \phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$$

Additive Synthesis

Beats

$$\sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$

$$\sin(\theta - \phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$$

so

$$\sin(\theta + \phi) + \sin(\theta - \phi) = 2\sin\theta\cos\phi$$

Additive Synthesis

Beats

$$\sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$

$$\sin(\theta - \phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$$

so

$$\sin(\theta + \phi) + \sin(\theta - \phi) = 2\sin\theta\cos\phi$$

so

$$\sin(\alpha) + \sin(\beta) = 2\sin\left(\frac{\alpha + \beta}{2}\right)\cos\left(\frac{\alpha - \beta}{2}\right)$$

```
class TwoSine1 implements AudioSignal {
  float fl, fr;
  float pl = 0, pr = 0;
  TwoSine1(float fleft, float fright) { fl = fleft; fr = fright; }
  void generate(float [] buf) {
    for (int i = 0; i < buf.length; i++) {
      buf[i] = sin(2*PI*fl*i / 44100 + pl);
      buf[i] += sin(2*PI*fr*i / 44100 + pr);
      buf[i] = buf[i]/2;
    }
    pl += 2*PI*fl*buf.length/44100;
    pr += 2*PI*fr*buf.length/44100;
    pl = pl - 2*PI*floor(pl/(2*PI));
    pr = pr - 2*PI*floor(pr/(2*PI));
  }
  void generate(float [] left, float [] right) {
    for (int i = 0; i < left.length; i++) {
      left[i] = sin(2*PI*fl*i / 44100 + pl);
      right[i] = sin(2*PI*fr*i / 44100 + pr);
    }
    pl += 2*PI*fl*left.length/44100;
    pr += 2*PI*fr*left.length/44100;
    pl = pl - 2*PI*floor(pl/(2*PI));
    pr = pr - 2*PI*floor(pr/(2*PI));
  }
}
```

```
class TwoSine implements AudioSignal {
  float fl, fr;
  float pl = 0, pr = 0;
  TwoSine(float fleft, float fright) { fl = fleft; fr = fright; }
  float sinebuf(float [] buf, float f, float a, float p) {
    return sinebuf(buf, f, a, p, true);
  }
  float sinebuf(float [] buf, float f, float a, float p, boolean zero) {
    for (int i = 0; i < buf.length; i++) {
      if (zero)
        buf[i] = 0;
      buf[i] += a*sin(2*PI*f*i / 44100 + p);
    }
    p += 2*PI*f*buf.length/44100;
    return p - 2*PI*floor(p/(2*PI));
  }
  void generate(float [] buf) {
    pl = sinebuf(buf, fl, 0.5, pl);
    pr = sinebuf(buf, fr, 0.5, pr, false);
  }
  void generate(float [] left, float [] right) {
    pl = sinebuf(left, fl, 1, pl);
    pr = sinebuf(right, fr, 1, pr);
  }
}
```