

# EVOLUSJONSALGORITMER

Optimering ved bruk av mekanismer  
inspirert av biologisk evolusjon.

Jørgen Høgberget

# HOVEDIDÉEN

## EN POPULASJON SOM UNDERGÅR...

- Seleksjon
- Reproduksjon
- Mutasjon

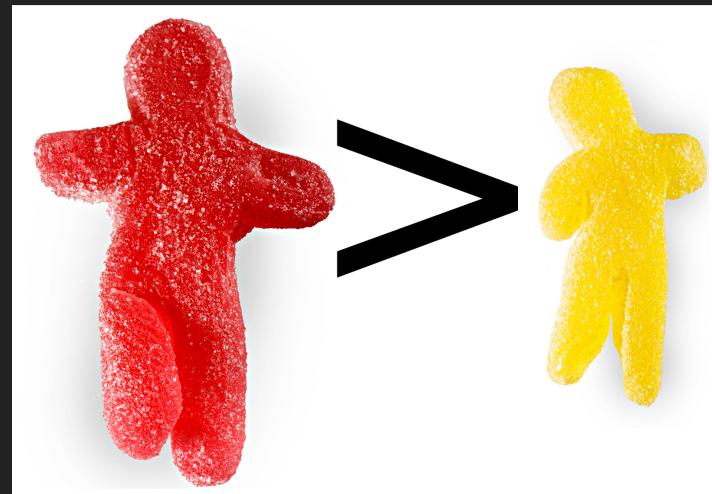
Dette er ofte også referert til som en genetisk algoritme.

# POPULASJONEN



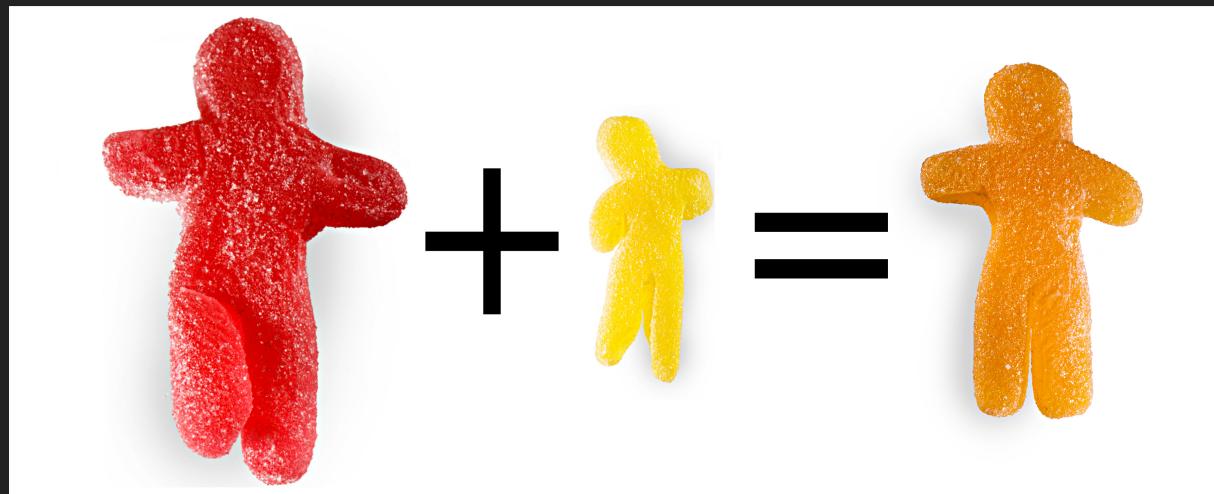
Består av en mengde individer, hvor et individ representerer en komplett mulig løsning på problemet.

# SELEKSJON



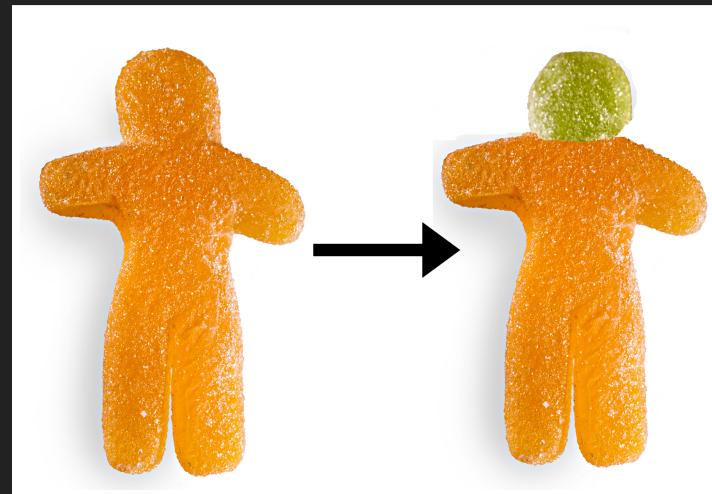
For å kunne velge det beste individet må vi kunne måle hvor bra det presterer. Dette gjøres ved bruk av en fit-funksjon.

# REPRODUKSJON



Individene som scorer best får reproduksjon.  
To individer kombineres til et nytt ved midling.  
Et individ med lav score erstattes av det nye individet.

# MUTASJON



For å kunne introdusere nye løsninger så vil alle avkom bli gitt en tilfeldig mutasjon.

# ABSTRAKT IMPLEMENTASJON

```
abstract class Population (sizec: Int) {  
    val size = sizec  
  
    def calculateFit(id: Int) : Float  
  
    def mutateIndividual(id: Int) : Unit  
  
    def reproduce(idParent1: Int, idParent2: Int, idChild: Int) : Unit  
    ...  
}
```

# IMPLEMENTASJON AV EVOLUSJON

```
def evolve(nMax: Int, convErr: Float) : Int = {
    val sortedIds = Array.range(0, size)
    val fits = Array.tabulate(size)(n => calculateFit(n))

    inplaceSort(sortedIds, fits)

    var n = 0
    while (fits(sortedIds(0)) > convErr & n < nMax) {

        for (i <- Range(1, size/2)) {
            val child = sortedIds(size - i)
            generateOffspring(sortedIds(0), sortedIds(i), child)
            mutateIndividual(child)
            fits(child) = calculateFit(child)
        }
    }
}
```

# EKSEMPEL: FOURIER-TRANSFORMER

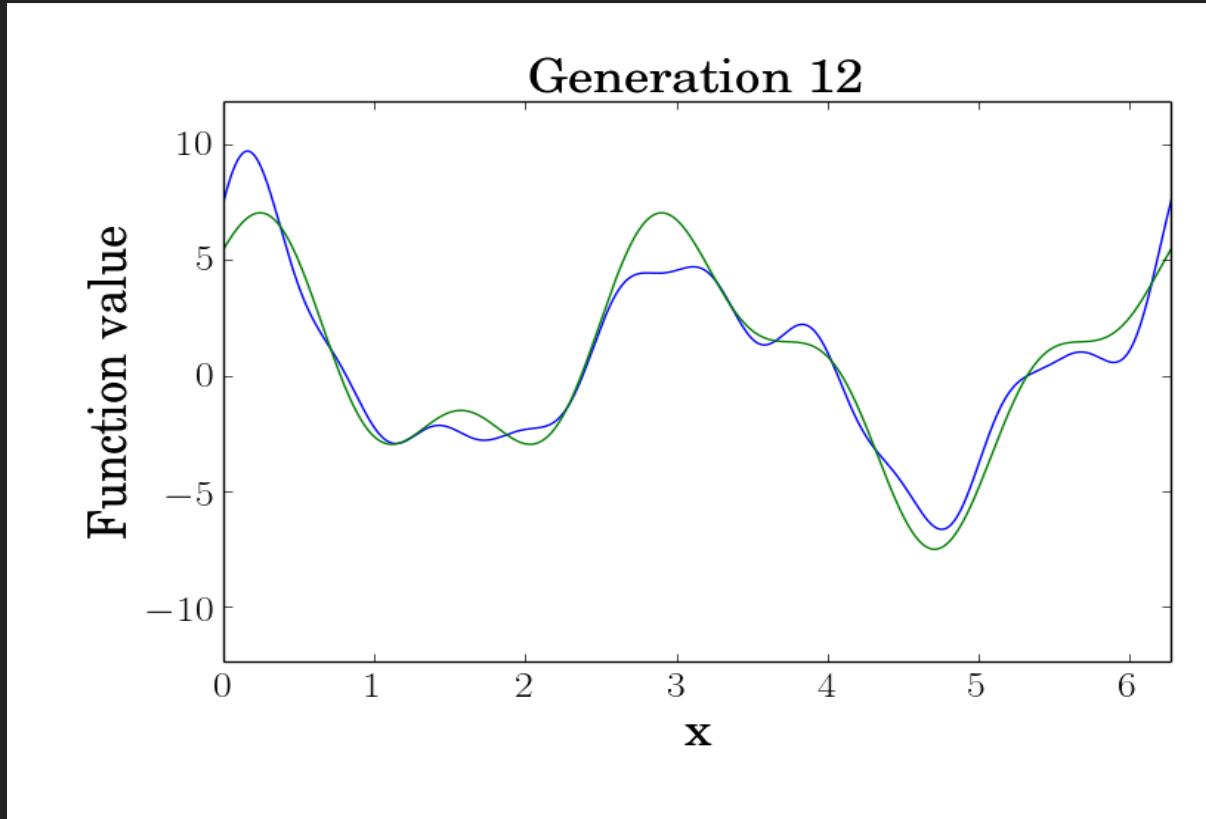


Et individ er en spesifikk sammensetning av bølger.

## EKSEMPEL: FOURIER-TRANSFORMER

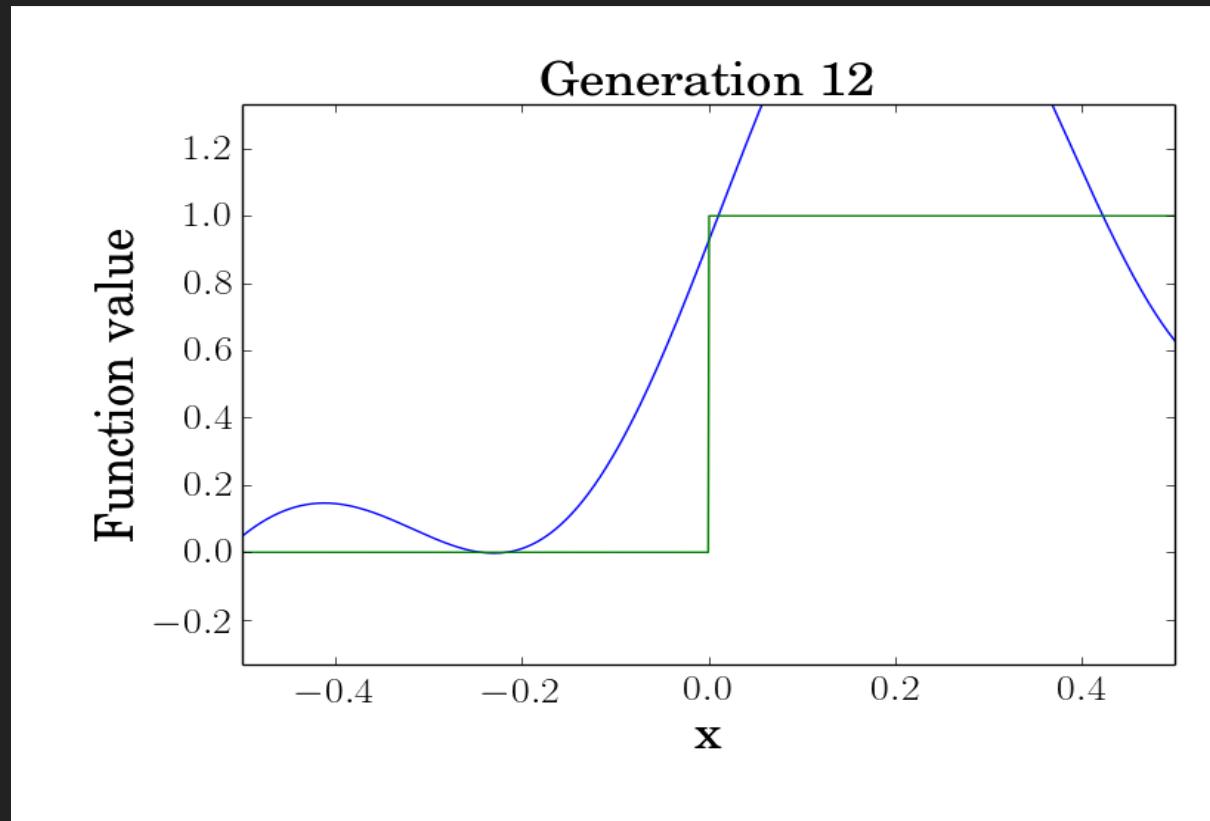
- Fit-funksjonen er avstanden mellom løsningen produsert av individet og den eksakte funksjonen.
- Reproduksjon skjer ved midling over foreldrenes bølger.
- Mutasjon skjer ved en tilfeldig endring i en tilfeldig bølge.

# ET TILFELLE MED EKSAKT LØSNING



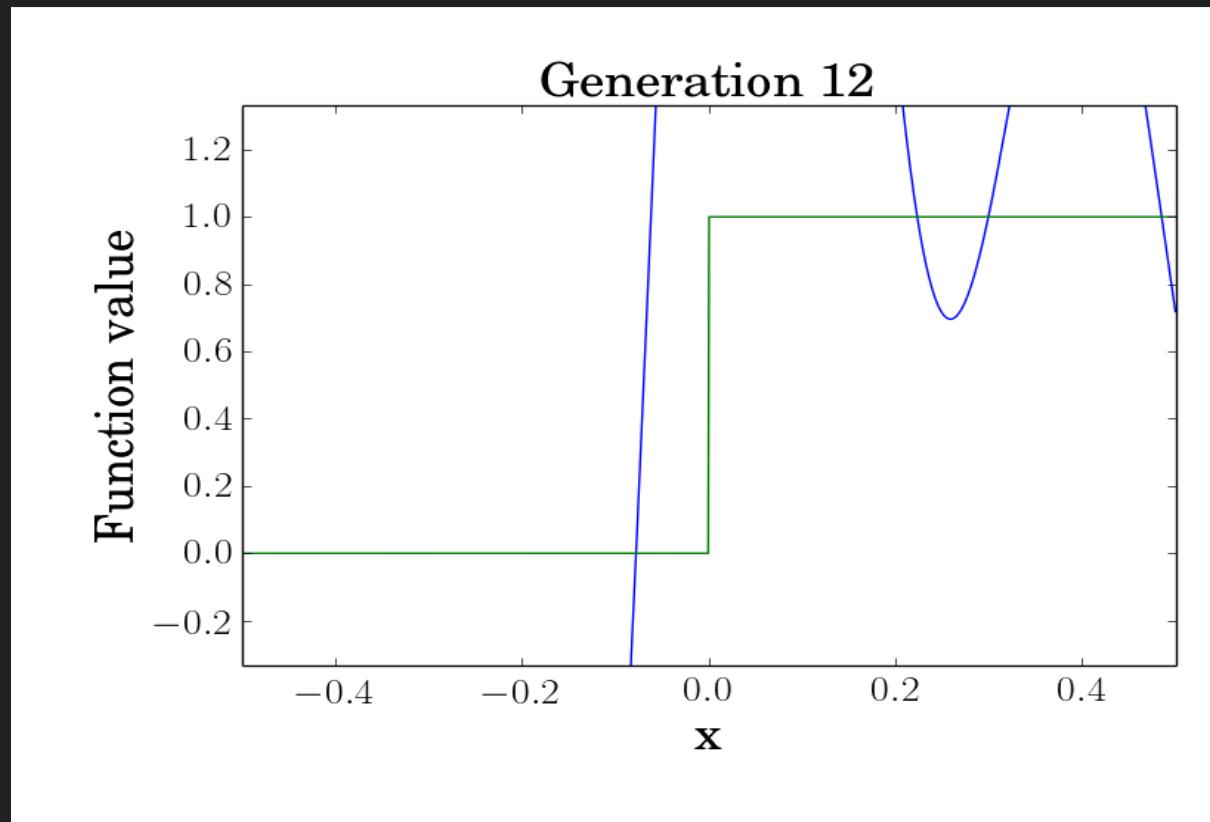
Et inputsignal er konstruert av tilgjengelige harmoniske bølger og blir derfor perfekt rekonstruert.

# ILDPROØVE



Et ikke-periodisk skarpt signal krever uendelig mange bølger. Her: 10.

# ILDPRØVE



Samme som forrige slides bare nå med 20 bølger.

# IMPLEMENTASJON AV FIT-FUNKSJON

```
final override def calculateFit(id: Int) : Float = {  
  (targetValues, calculateFourierFunction(id)).zipped  
    .map((f, g) => (f-g)*(f-g)).sum/targetValues.size  
}
```

```
def calculateFourierFunction(id: Int) : List[Float] = {  
  (cosines, sines).zipped.map((c, s) =>  
    a0(id) + fourierElement(cosineCoeffs(id), c) + fourierElement(sir  
  }
```

```
def fourierElement(coeffs: Array[Float], trigioms: List[Float]) : Flc  
  (coeffs, trigioms).zipped.map(_ * _).sum  
}
```

# IMPLEMENTASJON AV REPRODUKSJON

```
final override def reproduce(idParent1: Int, idParent2: Int, idChild: Int): Unit = {  
    a0(idChild) = (a0(idParent1) + a0(idParent2))/2  
  
    mixParents(sineCoeffs, idParent1, idParent2, idChild)  
    mixParents(cosineCoeffs, idParent1, idParent2, idChild)  
}
```

```
//50-50 mix of each parent  
def mixParents(coeffs: List[Array[Float]], idFirstParent: Int, idSecondParent: Int, idChild: Int): Unit = {  
    val ncoeffs = coeffs.length  
    val (coeffs1, coeffs2) = (coeffs(idFirstParent), coeffs(idSecondParent))  
    Range(0, ncoeffs).foreach(i => coeffs(idChild)(i) = (coeffs1(i) + coeffs2(i))/2)
```

# IMPLEMENTASJON AV MUTASJON

```
final override def mutateIndividual(id: Int) : Unit = {
    val coefficient = floor((2*ncoeffs+1)*Random.nextFloat()).toInt
    val changeFactor = 1.0f + Random.nextGaussian().toFloat

    if (coefficient == 0) {
        a0(id) = mutateSingle(a0(id), changeFactor)
    }
    else if (coefficient > ncoeffs) {
        val relCoeff = coefficient - ncoeffs - 1
        val coeff = cosineCoeffs(id)(relCoeff)
        cosineCoeffs(id)(relCoeff) = mutateSingle(coeff, changeFactor)
    }
    else {
        val relCoeff = coefficient - 1
        val coeff = sineCoeffs(id)(relCoeff)
        sineCoeffs(id)(relCoeff) = mutateSingle(coeff, changeFactor)
    }
}
```

```
def mutateSingle(coeff: Float, changeFactor: Float) : Float = {
    changeFactor*coeff
}
```

## FORBEDRINGSPOTENSIALER:

- Mer avansert reproduksjonsrett.
- Mer varierte mutasjoner (flere av gangen osv.).
- Kun kreativiteten som setter grenser.

# EVOLUSJONSALGORITMER ER GODE NÅ...

- løsningen har mange parametre.
- man lett kan måle hvor bra (relativt) en løsning er.
- det finnes en logisk måte å kombinere/mutere løsninger.
- den optimale løsningen endrer seg underveis.

**TAKK FOR OPPMERKSOMHETEN!**