Töluleg greining

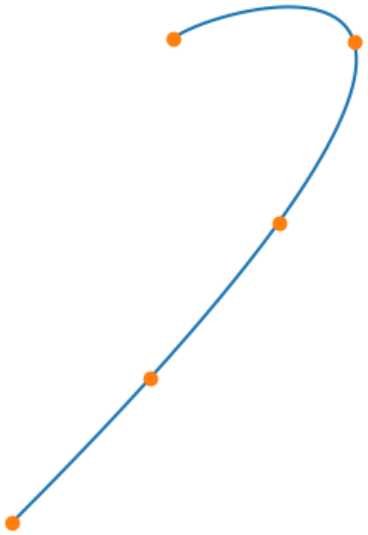
(STÆ405G vor 2019)

Umstikanir sléttuferla

Höfundar:
Álfheiður Edda Sigurðardóttir
Hannes Kristinn Árnason
Matthias Baldursson Harksen

Leiðbeinandi:
Sigurður Freyr Hafstein

25. mars 2019



1 Fræði

Í þessari skýrslu munum við skoða umstikanir á sléttuferlum. Lítum því á feril:

$$\alpha(t) := (x(t), y(t)), \qquad t \in [0,1]$$

Við skilgreinum lengd sléttuferilsins α með:

$$\ell_\alpha := \int_0^1 \sqrt{\langle \alpha'(t), \alpha'(t) \rangle} dt = \int_0^1 \sqrt{(x'(t))^2 + (y'(t))^2} dt.$$

Ef við lítum á sem svo að $\alpha(t)$ lýsi staðsetningu agnar í tvívíðu rúmi við tímann t. Þá er eðlilegt að skilgreina heildarvegalengdina sem ögnin hefur ferðast í sléttunni frá upphafspunkti sínum sem fall af T sem:

$$\ell_\alpha(T) := \int_0^T \sqrt{\langle \alpha'(t), \alpha'(t) \rangle} dt = \int_0^T \sqrt{(x'(t))^2 + (y'(t))^2} dt. \tag{1}$$

Við tölum þá um $\ell_\alpha(T)$ sem bogalengd ferilsins α frá 0 til T. Þá er $\ell_\alpha = \ell_\alpha(1)$, sér í lagi.

Einnig er eðlilegt að hugsa um $\alpha'(t) = (x'(t), y'(t))$ sem hraða agnarinnar hverrar staðsetningu $\alpha(t)$ lýsir og $\|\alpha'(t)\|$ sem ferð hennar. Ef við höfum $\|\alpha'(t)\| = k$ fyrir öll $t \in [0,1]$ og einhvern fasta k getum við því talað um að ögnin ferðist með jöfnum hraða. Jafngilt er að

$$\ell_\alpha(T) = \int_0^T \|\alpha'(t)\| dt = Tk$$

Ef við látum $s \in [0,1]$ tákna hlutfall þess hve langt við erum komin af lengdinni ℓ_α þá má finna ótvírætt $t^*(s) \in [0,1]$ þannig að:

$$\ell_\alpha(t^*(s)) = s\ell_\alpha. \tag{2}$$

Að leysa þessa jöfnu er jafngilt því að finna þjált gagntækt fall $\theta : [0,1] \to [0,1]$ þannig að $\|\alpha(\theta(t))\| = \ell_\alpha$ fyrir öll $t \in [0,1]$, samanber athuganir okkar að ofan. Við segjum þá að við höfum *umstikað* ferilinn α, enda gefum við nýja stikun sem breytir ekki mynd α.

Til þess að ákvarða $t^*(s)$ í jöfnu (2) fyrir gefið s þá er nóg að finna núllstöðvar fallsins:

$$f(T) := \int_0^T \sqrt{(x'(t))^2 + (y'(t))^2} dt - s\ell_\alpha, \qquad T \in [0,1]. \tag{3}$$

Það má gera með helmingunarleit eða með aðferð Newtons. Við athugum að $f(0) < 0$ (ef $s \neq 0$) en $f(1) > 0$ (ef $s \neq 1$) auk þess sem f er vaxandi fall svo við höfum nákvæmlega eina rót á bilinu $[0,1]$.
Athugum einnig að samkvæmt höfuðreglu diffur- og tegurreiknings fæst:

$$f'(T) = \sqrt{(x'(T))^2 + (y'(T))^2} = \|\alpha'(T)\|, \qquad T \in [0,1]. \tag{4}$$

Almennar gætum við viljað finna umstikun á α þannig að „framvindu“ agnarinnar eftir ferlinum sé lýst með gefnu vaxandi falli $c : [0,1] \to [0,1]$. Með öðrum orðum þannig að

$$\ell_\alpha(t^*(c(s))) = c(s)\ell_\alpha. \tag{5}$$

Þetta má leysa á sama hátt og jöfnu (2), við þurfum bara að ákvarða núllstöðvar fallsins

$$g(T) := \int_0^T \sqrt{(x'(t))^2 + (y'(t))^2} dt - c(s)\ell_\alpha, \qquad T \in [0,1]. \tag{6}$$

Athugum að lokum að lýsa má einföldum skilyrðum fyrir gefinn sléttuferil með jöfnum fyrir fallið t^*. Æskilegt gæti verið að skipta gefnum ferli í jafnlanga hlutferla. Það felur í sér að ákvarða stök $t_0, \ldots, t_{n+1} \in [0,1]$ þannig að

$$0 = t_0 < t_1 < \ldots < t_{n-1} < t_n = 1$$

og

$$\frac{\ell_\gamma}{n} = \int_{t_i}^{t_{i+1}} \sqrt{(x'(t))^2 + (y'(t))^2} dt = \ell_\gamma(t_{i+1}) - \ell_\gamma(t_i)$$

fyrir $i = 0, \ldots, n-1$. Við athugum þá að t_i er gefinn með

$$t_i = t^* \left(i \cdot \frac{\ell\gamma}{n} \right). \tag{7}$$

2 Útfærsla

(1) Við byrjum á því að skrifa fall sem metur lengd ferils frá $t = 0$ upp í $t = T$, þ.e.a.s. fall sem reiknar $\ell_\alpha(T)$ fyrir gefinn sléttuferil $\alpha(t)$. Til þess notum við aðhæfða tegrun (e. adaptive quadrature) sem byggir á trapisuaðferðinni. Okkar útfærsla reiknar lengd ferilsins milli punktanna a_0 og b_0, því heildið frá 0 upp í T er sértilfellið þegar $a_0 = 0$.

Aðhæfð tegrun með trapisunálgunum:

```
''' Finds the integral of f over [a0,b0]
within a tolerance tol0 via the Trapezoid method'''
def adaptiveQuadrature(f, a0, b0, tol0):
sum = 0
n = 1
a = np.zeros(1000)
b = np.zeros(1000)
app = np.zeros(1000)
tol = np.zeros(1000)
a[0] = a0
b[0] = b0
tol[0] = tol0
app[0] = Trap(f, a0, b0)
while n > 0:
    if len(app) < 2*n:
        app = lengthen(app)
        a = lengthen(a)
        b = lengthen(b)
        tol = lengthen(tol)
    c = (a[n-1] + b[n-1])/2
    oldapp = app[n-1]
    app[n-1] = Trap(f, a[n-1], c)
    app[n] = Trap(f, c, b[n-1])
    if np.abs(oldapp-app[n-1]-app[n]) < 15*tol[n-1]:
        sum += app[n-1]+app[n]
        n -= 1
    else:
        b[n] = b[n-1]
        b[n-1] = c
        a[n] = c
        tol[n-1] /=2
        tol[n] = tol[n-1]
```

```
            n += 1
    return sum
```

þar sem við höfum notað eftirfarandi hjálparföll:

```
'''Returns the trapzoid approximation of the integral of f over [a,b]'''
def Trap(f, a, b):
    return (f(a)+f(b))*(b-a)/2

''' Doubles the length of the vector v when needed (while storing the values)'''
def lengthen(v):
    m = len(v)
    newv = np.zeros(2*m)
    for i in range(m):
        newv[i]=v[i]
    return newv
```

Einnig betrumbættum við aðferðinna til þess að nota Simpson á hverju bili í von um betri keyrslutíma. Við sýnum þá útfærslu hér:

Aðhæfð tegrun með nálgun Simpsons:

```
''' Finds the integral of f over [a0,b0]
within a tolerance tol0 via the Simpson method'''
def adaptiveSimpsonsQuadrature(f, a0, b0, tol0):
    sum = 0
    n = 1
    a = np.zeros(1000)
    b = np.zeros(1000)
    app = np.zeros(1000)
    tol = np.zeros(1000)
    a[0] = a0
    b[0] = b0
    tol[0] = tol0
    app[0] = Simpson(f, a0, b0)
    while n > 0:
        if len(app) < 2*n:
            app = lengthen(app)
            a = lengthen(a)
            b = lengthen(b)
            tol = lengthen(tol)
        c = (a[n-1] + b[n-1])/2
        oldapp = app[n-1]
        app[n-1] = Simpson(f, a[n-1], c)
        app[n] = Simpson(f, c, b[n-1])
        if np.abs(oldapp-app[n-1]-app[n]) < 3*tol[n-1]:
            sum += app[n-1]+app[n]
            n -= 1
        else:
            b[n] = b[n-1]
            b[n-1] = c
            a[n] = c
            tol[n-1] /=2
            tol[n] = tol[n-1]
            n += 1
    return sum
```

þar sem við höfum notað eftirfarandi hjálparfall:

```
'''Returns the Simpson approximation of the integral of f over [a,b]'''
def Simpson(f,a,b):
    return (f(a)+f(b)+4*f((a+b)/2))*(b-a)/6
```

Ef P er nú ferillinn gefinn með $P(t) = (x(t), y(t))$ þar sem

$$x(t) = 0,5 + 0,3t + 3,9t^2 + 4,7t^3, \quad y(t) = 1,5 + 0,3t + 0,9t^2 + 2,7t^3$$

þá gefa eftirfarandi fallsköll lengdina ℓ_γ.

```
x = lambda t: 0.5 + 0.3*t + 3.9*t**2 - 4.7*t**3
y = lambda t: 1.5 + 0.3*t + 0.9*t**2 - 2.7*t**3
dx = lambda t: 0.3 + 7.8*t - 14.1*t**2
dy = lambda t: 0.3 + 1.8*t - 8.1*t**2
arc = lambda t: np.sqrt(dx(t)**2 + dy(t)**2)

adaptiveQuadrature(arc, 0, 1, 0.00001)
Out[1]: 2.495253347469116
```

(2) Nú skrifum við fall sem reiknar $t^*(s)$ og leysir þannig jöfnu (2). Jafngilt er að finna rót fallsinns f í jöfnuni og til þess nýtum við okkur helmingunaraðferð sem útfærist svo:

```
''' Finds the root of f betwen [a,b] up to a an interval [c,d] of length tol.'''
def binary(f, a, b, tol):
    while (b-a)/2 > tol:
        c = (a+b)/2
        if f(c) == 0:
            return c
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return c
```

og að því gefnu er einfalt að skrifa fall sem ákvarðar $t^*(s)$ sem rót f fyrir gefið s. Það fall sýnum við hér að neðan:

```
'''Finds the value of tstar via binary method (which is defined in the text)'''
def tstar(arc, s, tollength, tolbinary):
    l = adaptiveQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveQuadrature(arc,0,t,tollength)-l*s
    return binary(Arc, 0, 1, tolbinary)
```

Líkt og við athuguðum í kafla 1 þá hefur Arc nákvæmlega eina rót á bilinu $[0, 1]$ sem binary finnur með umbeðinni nákvæmni. Þetta fall var prófað fyrir gildið $s = 0.7$ með eftirfarandi útkomu.

```
s = 0.7
tstarr = tstar(arc, s, TOL, TOL)
print("The value of t*("+ str(s) + ") is: "+str(tstarr))

Out[1]: The value of t*(0.7) is: 0.8971652984619141
```

(3) Nú skiptum við sléttuferlinum P sem við skilgreindum í lið (1) niður í n jafnlanga hlutferla fyrir gildin $n = 4$ og $n = 20$. Við gefum nú fall sem finnur þessa skiptingu út frá jöfnu 7:

```
'''Equipartitions the arc given'''
def equiPartition(arc, n, tol):
    partition = [ tstar(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition
```

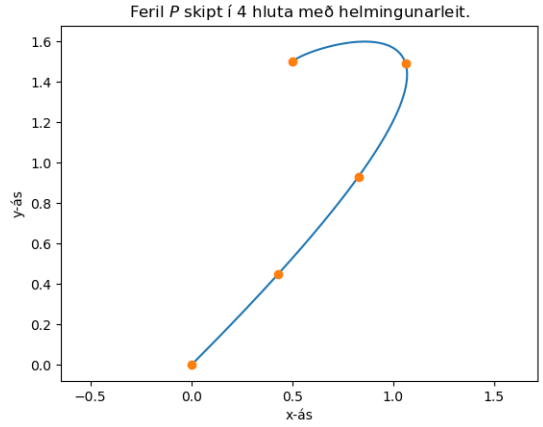
Fall sem tekur inn lista af skiptipunktum ásamt stikaframsetningu $P = (x, y)$ og teiknar ferilinn og skiptingu hans er útfært á eftirfarandi hátt:

```
'''Plots the partition given'''
def plotCurvePartition(x, y, partition, label):
    t = np.linspace(0, 1, 500)
    vx = [x(s) for s in t]
    vy = [y(s) for s in t]
    xpoint = [x(s) for s in partition]
    ypoint = [y(s) for s in partition]

    plt.figure()
    plt.plot(vx, vy)
    plt.plot(xpoint, ypoint, 'o')
    plt.axis("equal")
    plt.title(label)
    plt.xlabel("x-ás")
    plt.ylabel("y-ás")
    plt.show()
return
```

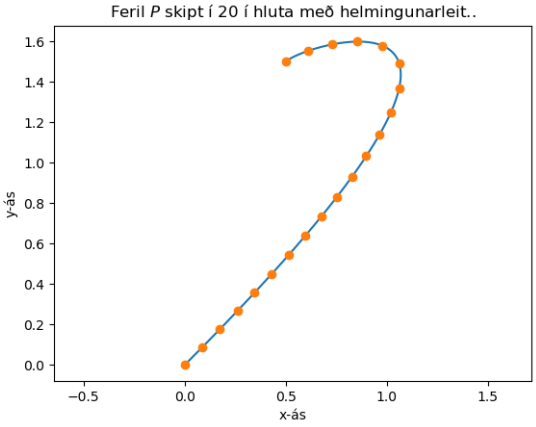
Fallsköllin

```
    partition4, partition20 = equiPartition(arc, 4), equiPartition(arc, 20)
    plotCurvePartition(x, y, equiPartition(arc, 4, tol),
                       "Feril $P$ skipt í 4 hluta með helmingunarleit.")
    plotCurvePartition(x, y, equiPartition(arc, 20, tol),
                       "Feril $P$ skipt í 20 í hluta með helmingunarleit..")
```

gefa þá eftirfarandi myndir af skiptingum sléttuferilsins.



Mynd 1: Teikning af skiptingu sléttuferilsins P fyrir $n = 4$, þar sem skiptipunkar voru fundnir með helmingunarleit og trapisuaðferð.



Mynd 2: Teikning af skiptingu sléttuferilsins P fyrir $n = 20$, þar sem skiptipunkar voru fundnir með helmingunarleit og trapisuaðferð.

(4) Nú má búast við því að unnt sé að bæta keyrslutíma fallsins okkar `tstar` í lið (2) með því að skipta helmingunarleit út fyrir aðferð Newtons eða skipta út aðhæfðri heildun með trapisuaðferð út fyrir heildun með aðferð Simpsons. Við útfærum því aðferð Newtons ásamt tveimur föllum til viðbótar en samanber jöfnu (4) þá er afleiða fallsins f í jöfnu (3) gefin með fallinu arc er við skilgreindum áður.

```
'''Finds the root of f closes to x0 with Newtons method'''
def Newton(f,df,x0,tol):
    x = x0
    while np.abs(f(x)) > tol:
        x -= f(x)/df(x)
return x

'''Finds the value of tstar via Newton'''
def tstar2(arc, s, tollength, tolNewton):
    l = adaptiveQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveQuadrature(arc,0,t,tollength)-l*s
    return Newton(Arc,arc, s, tolNewton)

'''Finds the value of tstar via Newton with Simpson AQ integration'''
def tstar3(arc, s, tollength, tolNewton):
    l = adaptiveSimpsonsQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveSimpsonsQuadrature(arc,0,t,tollength)-l*s
    return Newton(Arc,arc, s, tolNewton)
```

Einnig gefum við tímamælingafall

```
'''Times the length of calling the function a given number of times.'''
def timeFunction(func, *args, **kwargs):
    def wrapped():
        return func(*args)
    # timeit takes keyword number to control number of iterations
    return timeit.timeit(wrapped, **kwargs)
```

Eftirfarandi fallsköll bera þá saman útkomur fallanna þriggja ásamt keyrslutímum þeirra fyrir 10 keyrslur.

```
s = 0.7
tol = 0.0001
TOL = 0.000001

tstarr  = tstar(arc, s, TOL, TOL)
tstarr2 = tstar2(arc, s, TOL, TOL)
tstarr3 = tstar3(arc, s, TOL, TOL)
print("The value of t*("+ str(s) + ") is: "+str(tstarr))
print("The value of t*("+ str(s) + ")
      computed by Newton's method is: "+str(tstarr))
print("The value of t*("+ str(s) + ")
      computed by Newton's method is: "+str(tstarr))

s=0.5
print("Time of computing t* with AQ trapizoid and bisection method:")
print(timeFunction(tstar, arc, s, tol, tol, number=10))

print("Time of computing t*2 with AQ trapizoid and Newtons method:")
print(timeFunction(tstar2, arc, s, tol, tol, number=10))
```

```
print("Time of computing t*3 with AQ Simpson and Newtons method:")
print(timeFunction(tstar3, arc, 0.5, tol, tol, number=10))
```

Útkoman verður

```
The value of t*(0.7) is: 0.8971652984619141
The value of t*(0.7) computed by Newton's method is: 0.8971657427381851
The value of t*(0.7) computed by Newton's method is: 0.8971657657478878
Time of computing t* with AQ trapizoid and bisection method:
0.9675356999996438
Time of computing t*2 with AQ trapizoid and Newtons method:
0.3302222000002075
Time of computing t*3 with AQ Simpson and Newtons method:
0.09078059999956167
```

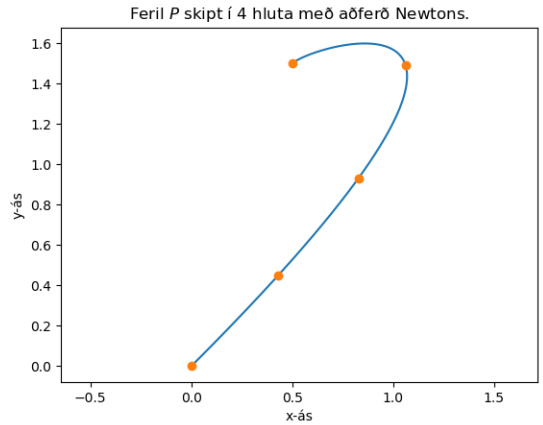
og sjáum við að aðferð Newtons bætir keyrslutímann töluvert en aðferð Simpsons styttir hann enn meira. Einnig fáum við sömu útkomu og með umbeðinni nákvæmni í lið (2)

Einfalt er þá að breyta skiptifallinu til þess að nota annars vegar `tstar2` og hins vegar `tstar3` sem gefur þá samsvarandi hlutfallslega bætingu keyrslutíma, þar sem fallið gerir ekkert nema að kalla á tstar $n+1$ sinnum.

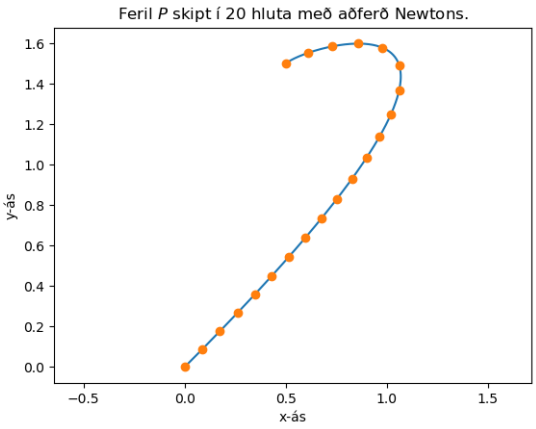
```
'''Equipartitions the arc given'''
def equiPartition2(arc, n, tol):
    partition = [ tstar2(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition

'''Equipartitions the arc given'''
def equiPartition3(arc, n, tol):
    partition = [ tstar3(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition
```

Eftirfarandi fallsköll gefa svipaðar myndir og í lið (3) en við sýnum þær að neðan.

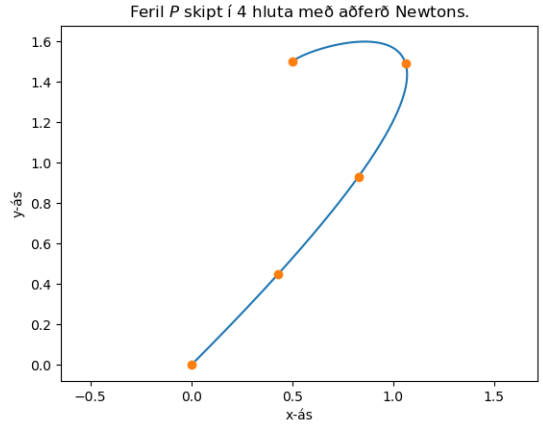
```
plotCurvePartition(x, y, equiPartition2(arc, 4, tol),
                   "Feril $P$ skipt í 4 hluta með aðferð Newtons.")
plotCurvePartition(x, y, equiPartition2(arc, 20, tol),
                   "Feril $P$ skipt í 20 hluta með aðferð Newtons.")

plotCurvePartition(x, y, equiPartition3(arc, 4, tol),
                   "Feril $P$ skipt í 4 hluta með aðferð Newtons.")
plotCurvePartition(x, y, equiPartition3(arc, 20, tol),
                   "Feril $P$ skipt í 20 hluta með aðferð Newtons.")
```



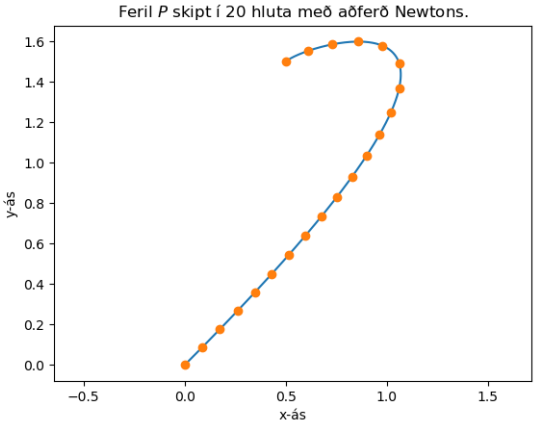
Mynd 3: Teikning af skiptingu sléttuferilsins P fyrir $n = 4$, þar sem skiptipunkar voru fundnir með aðferð Newtons og trapisuaðferð.



Mynd 4: Teikning af skiptingu sléttuferilsins P fyrir $n = 20$, þar sem skiptipunkar voru fundnir með aðferð Newtons og trapisuaðferð.



Mynd 5: Teikning af skiptingu sléttuferilsins P fyrir $n = 4$, þar sem skiptipunkar voru fundnir með aðferð Newtons og aðferð Simpsons.



Mynd 6: Teikning af skiptingu sléttuferilsins P fyrir $n = 20$, þar sem skiptipunkar voru fundnir með aðferð Newtons og aðferð Simpsons.

(5) Skrifum nú fall sem sýnir hreyfingu punkts eftir ferlinum á tíma t að gefinni umstikun $s(t)$ á ferlinum.

```
'''Updates the point, used in the animation method.'''
def updatePoint(n, x, y, s, point):
    point.set_data(np.array([x(s[n]), y(s[n])]))
    return point,

'''Animates the curve.'''
def animateCurve(x, y, s, label):
    fig = plt.figure()
    # create the underlying path
    t = np.linspace(0, 1, 500)
```

```
        vx, vy = x(t), y(t)
        line, = plt.plot(vx, vy)
        # plot the first point
        point, = plt.plot([vx[0]], [vy[0]], 'o')

        plt.title(label)
        plt.xlabel("x-ás")
        plt.ylabel("y-ás")
        plt.axis('equal')

        ani=animation.FuncAnimation(fig, updatePoint, len(s), fargs=(x, y, s, point), blit=True, in

    return ani
```

Fallskallið

```
ani1 = animateCurve(x, y, np.linspace(0,1,200), "Ferillinn $P$ stikaður af $t$.")
ani1.save("ani1.mp4", writer="ffmpeg", fps=30)
```

gefa þá hreyfimynd er sýnir hreyfingu punktsins $\gamma(t)$ sem fall af t en fallsköllin

```
sVec = [tstar3(arc,t,tol,tol) for t in np.linspace(0,1,200)]
ani2 = animateCurve(x, y, sVec, "Ferillinn $P$ stikaður af bogalengd.")
ani2.save("ani2.mp4", writer="ffmpeg", fps=30)
```

sýna hreyfingu með jöfnum hraða í samræmi við athuganir okkar í fræða kaflanum.

Hægt er að sjá hreyfimyndirnar á eftirfarandi vefslóðum:

- `https://giphy.com/channel/mbh6`
- `https://github.com/mrharksen/motionControl/tree/master/videos`

(6) Nú prófum við aðferðir okkar á ferlum að eigin vali. Þá ferla viljum við einnig stika af bogalengd og útbúa hreyfimynd af ferðalagi eftir ferlunum.
Fyrst beitum við aðferðunum á Bézier splæsiferilinn um punktana $(0, 0)$ og $(0, 1)$ með stýripunktana $(-1, 1)$ og $(0, 2)$. Hann smíðum við og mælum lengd hans með eftirfarandi línum:

```
p1 = (0,0); p2 = (-1,1); p3 = (0,2); p4 = (0,1)
a1, a2, da1, da2 = Bezier(p1,p2,p3,p4)
arcBezier =  lambda t: np.sqrt(da1(t)**2 + da2(t)**2)
lBezSimp = adaptiveSimpsonsQuadrature(arcBezier, 0, 1, TOL)
print("The length of the Bézier curve is: "+ str(lBezSimp))
```

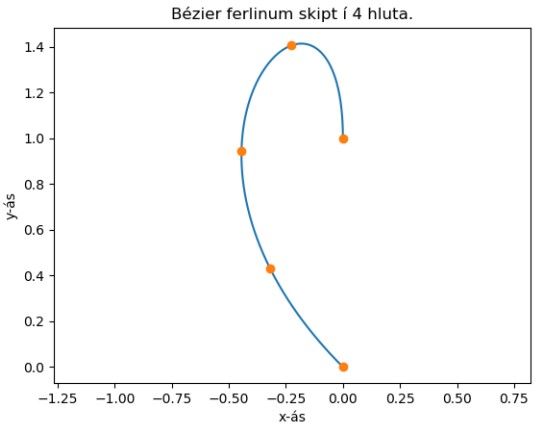
Við skiptum þessum ferli í annars vegar í 4 jafnlanga búta, og hins vegar 20, rétt eins gert var við ferilinn P.

```
plotCurvePartition(a1, a2, equiPartition3(arcBezier, 4, tol),
                   "Bézier ferlinum skipt í 4 hluta.")
plotCurvePartition(a1, a2, equiPartition3(arcBezier, 20, tol),
                   "Bézier ferlinum skipt í 20 hluta.")
```

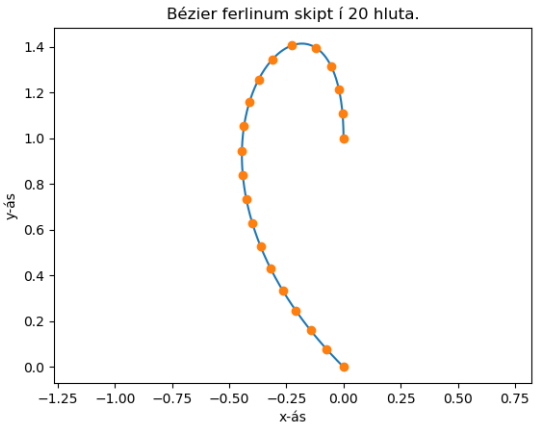
Til gamans gerðum við einnig hreyfimyndir eins og í lið (5) fyrir Bézier ferilinn, eina sem sýnir Bézier ferilinn teiknaðan eftir stikun hans en hin sýnir hann stikaðan af bogalengd. Hreyfimyndin fékkst með eftirfarandi fallsköllum:

```
aniBez1 = animateCurve(a1, a2, np.linspace(0,1,200),
                       "Bézier ferill stikaður af $t$.")
aniBez1.save("aniBez1.mp4", writer="ffmpeg", fps=30)
```

```
sVecBezier = [tstar3(arcBezier,t,tol,tol) for t in np.linspace(0,1,200)]
aniBez2 = animateCurve(a1, a2, sVecBezier,
                       "Bézier ferill stikaður af bogalengd.")
aniBez2.save("aniBez2.mp4", writer="ffmpeg", fps=30)
```



Mynd 7: Teikning af skiptingu Bézier ferils fyrir $n = 4$. Reiknað var með aðferð Newtons og aðferð Simpsons.



Mynd 8: Teikning af skiptingu Bézier feril fyrir $n = 20$. Reiknað var með aðferð Newtons og aðferð Simpsons.

(7) Látum nú $c : [0,1] \to [0,1]$ vera gefið vaxandi fall. Í samræmi við jöfnur 5 (6) má nota eitthvert fallanna okkar `tstar`, `tstar2` eða `tstar3` til þess að ákvarða gildið $t^*(c(s))$.

Fallið okkar animateFunction í lið (5) tekur inn stika s sem ákvarðar hreyfingu punktsins. Við getum því, fyrir gefna fallið c látið s vera gildi $(t^* \circ c)$ í jafndreifðum punktum $s_1, \ldots, s_n$ á $[0,1]$. Gefinn rammi hreyfimyndarinnar númer i sýnir þá stöðu agnarinnar í $\alpha(c(s_i))$ og við útfærum fall sem býr til umbeðinn vigur að falli c gefnu.

```
''' Finds the new progress speed for a given function c '''
def parametersFromProgressCurve(arc, c, n):
    return [tstar3(arc,c(t),tol,tol) for t in np.linspace(0,1,n)]
```

Nú getum við búið til samsvarandi hreyfimyndir og í lið (5) þar sem framvindu agnarinnar er lýst með föllunum

$$s \longmapsto s^{\frac{1}{3}}, \quad s \longmapsto s^2, \quad s \longmapsto \sin\left(s\frac{\pi}{2}\right) \quad \text{og} \quad s \longmapsto \frac{1}{2} + \frac{1}{2}\sin\left((2s-1)\frac{\pi}{2}\right).$$

Er þetta gert með eftirfarandi skipunum.

```
ct1d3 = lambda t: np.power(t,1/3)
st1d3 = parametersFromProgressCurve(arc,ct1d3,300)
anit1d3 = animateCurve(x,y,st1d3,
                       "Ferill $P$ umstikaður af $C(s)=s^\\frac{1}{3}$.")
anit1d3.save("anit1d3.mp4", writer="ffmpeg", fps=30)

ct2 = lambda t: np.power(t,2)
st2 = parametersFromProgressCurve(arc,ct2,150)
anit2 = animateCurve(x,y,st2,
                     "Ferill $P$ umstikaður af $C(s)=s^2$.")
anit2.save("anit2.mp4", writer="ffmpeg", fps=30)

cSin = lambda t: np.sin(t*np.pi/2)
```

```
sSin = parametersFromProgressCurve(arc,cSin,150)
aSin = animateCurve(x,y,sSin,
                    "Ferill $P$ umstikaður af $C(s)=\\sin(s\\pi/2)$.")
aSin.save("aniSin.mp4", writer="ffmpeg", fps=30)

cSin2 = lambda t: 1/2+1/2*np.sin((2*t-1)*np.pi/2)
sSin2 = parametersFromProgressCurve(arc,cSin2,150)
aSin2 = animateCurve(x,y,sSin2, "Ferill $P$ umstikaður af
                    $C(s)=\\frac{1}{2}+\\frac{1}{2}\\sin((2s-1)\\pi/2 ) $.")
aSin2.save("aniSin.mp4", writer="ffmpeg", fps=30)
```

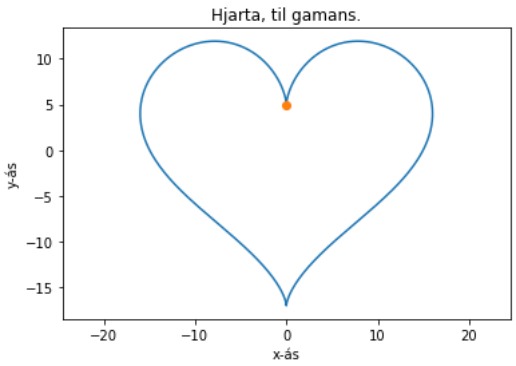
Punktafjöldi er hafður tvöfaldur fyrir fallið $s \mapsto s^{\frac{1}{3}}$ þar sem framvindan verður mjög hröð fyrir lítil gildi á s.

(8) Aukaliður: Þetta er stikun á hjartalaga ferli:

$$x(t) = 16\sin^3(2\pi t)$$
$$y(t) = 13\cos(2\pi t) - 5\cos(4\pi t) - 2\cos(6\pi t) - \cos(8\pi t).$$

Við gerðum hreyfimynd af þessu hjarta stikuðu af bogalengd.

```
Hx = lambda t: 16*(np.sin(2*np.pi*t))**3
Hy = lambda t: 13*np.cos(2*np.pi*t)-5*np.cos(4*np.pi*t)
                                   -2*np.cos(6*np.pi*t)-np.cos(8*np.pi*t)
Hdx = lambda t: 48*((np.sin(2*np.pi*t))**2)*np.cos(2*np.pi*t)
Hdy = lambda t: -13*np.sin(2*np.pi*t)+10*np.sin(4*np.pi*t)
                                     +6*np.sin(6*np.pi*t)+4*np.cos(8*np.pi*t)
Harc = lambda t: np.sqrt(Hdx(2*np.pi*t)**2 + Hdy(2*np.pi*t)**2)

heart = animateCurve(Hx, Hy, np.linspace(0,1,200), "Hjarta, til gamans.")
heart.save("heart.mp4", writer="ffmpeg", fps=30)
```



3 Kóði

Kóðann ásamt helstu teikningum má finna á Github: `https://github.com/mrharksen/motionControl`.

Hér má síðan finna gifin: `https://giphy.com/channel/mbh6`.

Hreyfimyndirnar á mp4 formi: `https://github.com/mrharksen/motionControl/tree/master/videos`

En við birtum einnig allan kóðann okkar hér fyrir neðan:

```
import timeit
import numpy as np
import numpy.linalg as lin
import matplotlib.pyplot as plt
from matplotlib import animation

'''
In order for this file to run ffmpeg is needed.
It is used to save the animations to an mp4 file.
If you do not want to install ffmpeg you can comment
out all sections where it says save().
'''

'''Returns the trapzoid approximation of the integral of f over [a,b]'''
def Trap(f, a, b):
    return (f(a)+f(b))*(b-a)/2

'''Returns the Simpson approximation of the integral of f over [a,b]'''
def Simpson(f,a,b):
    return (f(a)+f(b)+4*f((a+b)/2))*(b-a)/6

'''Returns the parametirization of the Bezier curve
which is defined by the points p1, p2, p3 and p4'''
def Bezier(p1,p2,p3,p4):
    bx = 3*(p2[0]-p1[0])
    cx = 3*(p3[0]-p2[0])-bx
    dx = p4[0] - p1[0] - bx - cx
    by = 3*(p2[1]-p1[1])
    cy = 3*(p3[1]-p2[1])-by
    dy = p4[1] - p1[1] - by - cy
    a1 = lambda t: p1[0] + bx*t + cx*t**2 + dx*t**3
    a2 = lambda t: p1[1] + by*t + cy*t**2 + dy*t**3
    da1 = lambda t: bx + 2*cx*t + 3*dx*t**2
    da2 = lambda t: by + 2*cy*t + 3*dy*t**2
    return a1, a2, da1, da2

''' Doubles the length of the vector v when needed (while storing the values)'''
def lengthen(v):
    m = len(v)
    newv = np.zeros(2*m)
    for i in range(m):
        newv[i]=v[i]
    return newv
```

```
''' Finds the integral of f over [a0,b0]
within a tolerance tol0 via the Trapezoid method'''
def adaptiveQuadrature(f, a0, b0, tol0):
    sum = 0
    n = 1
    a = np.zeros(1000)
    b = np.zeros(1000)
    app = np.zeros(1000)
    tol = np.zeros(1000)
    a[0] = a0
    b[0] = b0
    tol[0] = tol0
    app[0] = Trap(f, a0, b0)
    while n > 0:
        if len(app) < 2*n:
            app = lengthen(app)
            a = lengthen(a)
            b = lengthen(b)
            tol = lengthen(tol)
        c = (a[n-1] + b[n-1])/2
        oldapp = app[n-1]
        app[n-1] = Trap(f, a[n-1], c)
        app[n] = Trap(f, c, b[n-1])
        if np.abs(oldapp-app[n-1]-app[n]) < 15*tol[n-1]:
            sum += app[n-1]+app[n]
            n -= 1
        else:
            b[n] = b[n-1]
            b[n-1] = c
            a[n] = c
            tol[n-1] /=2
            tol[n] = tol[n-1]
            n += 1
    return sum

''' Finds the integral of f over [a0,b0]
within a tolerance tol0 via the Simpson method'''
def adaptiveSimpsonsQuadrature(f, a0, b0, tol0):
    sum = 0
    n = 1
    a = np.zeros(1000)
    b = np.zeros(1000)
    app = np.zeros(1000)
    tol = np.zeros(1000)
    a[0] = a0
    b[0] = b0
    tol[0] = tol0
    app[0] = Simpson(f, a0, b0)
    while n > 0:
        if len(app) < 2*n:
            app = lengthen(app)
            a = lengthen(a)
            b = lengthen(b)
```

```
            tol = lengthen(tol)
        c = (a[n-1] + b[n-1])/2
        oldapp = app[n-1]
        app[n-1] = Simpson(f, a[n-1], c)
        app[n] = Simpson(f, c, b[n-1])
        if np.abs(oldapp-app[n-1]-app[n]) < 3*tol[n-1]:
            sum += app[n-1]+app[n]
            n -= 1
        else:
            b[n] = b[n-1]
            b[n-1] = c
            a[n] = c
            tol[n-1] /=2
            tol[n] = tol[n-1]
            n += 1
    return sum

''' Finds the root of f betwen [a,b]
up to a an interval [c,d] of length tol.'''
def binary(f, a, b, tol):
    if np.abs(f(a)) < tol:
        return a
    if np.abs(f(b)) < tol:
        return b
    while (b-a)/2 > tol:
        c = (a+b)/2
        if f(c) == 0:
            return c
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return c

'''Finds the root of f closes to x0 with Newtons method'''
def Newton(f,df,x0,tol):
    x = x0
    while np.abs(f(x)) > tol:
        x -= f(x)/df(x)
    return x

'''Finds the value of tstar via binary method (which is defined in the text)'''
def tstar(arc, s, tollength, tolbinary):
    l = adaptiveQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveQuadrature(arc,0,t,tollength)-l*s
    return binary(Arc, 0, 1, tolbinary)

'''Equipartitions the arc given'''
def equiPartition(arc, n, tol):
    partition = [ tstar(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition

'''Finds the value of tstar via Newton'''
def tstar2(arc, s, tollength, tolNewton):
```

```
    l = adaptiveQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveQuadrature(arc,0,t,tollength)-l*s
    return Newton(Arc,arc, s, tolNewton)

'''Equipartitions the arc given'''
def equiPartition2(arc, n, tol):
    partition = [ tstar2(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition

'''Finds the value of tstar via Newton with Simpson AQ integration'''
def tstar3(arc, s, tollength, tolNewton):
    l = adaptiveSimpsonsQuadrature(arc, 0, 1, tollength)
    Arc = lambda t: adaptiveSimpsonsQuadrature(arc,0,t,tollength)-l*s
    return Newton(Arc,arc, s, tolNewton)

'''Equipartitions the arc given'''
def equiPartition3(arc, n, tol):
    partition = [ tstar3(arc, t, tol, tol) for t in np.linspace(0,1,n+1) ]
    return partition

'''Plots the partition given'''
def plotCurvePartition(x, y, partition, label):
    t = np.linspace(0, 1, 500)
    vx = [x(s) for s in t]
    vy = [y(s) for s in t]
    xpoint = [x(s) for s in partition]
    ypoint = [y(s) for s in partition]

    plt.figure()
    plt.plot(vx, vy)
    plt.plot(xpoint, ypoint, 'o')
    plt.axis("equal")
    plt.title(label)
    plt.xlabel("x-ás")
    plt.ylabel("y-ás")
    plt.show()
    return


'''Updates the point, used in the animation method.'''
def updatePoint(n, x, y, s, point):
    point.set_data(np.array([x(s[n]), y(s[n])]))
    return point,

'''Animates the curve.'''
def animateCurve(x, y, s, label):
    fig = plt.figure()
    # create the underlying path
    t = np.linspace(0, 1, 500)
    vx, vy = x(t), y(t)
    line, = plt.plot(vx, vy)
    # plot the first point
    point, = plt.plot([vx[0]], [vy[0]], 'o')
```

```
    plt.title(label)
    plt.xlabel("x-ás")
    plt.ylabel("y-ás")
    plt.axis('equal')

    ani=animation.FuncAnimation(fig, updatePoint, len(s), fargs=(x, y, s, point), blit=True, interval=25

    return ani

'''Times the length of calling the function ten times.'''
def timeFunction(func, *args, **kwargs):
    def wrapped():
        return func(*args)
    return timeit.timeit(wrapped, **kwargs)

''' Finds the new progress speed for a given function c '''
def parametersFromProgressCurve(arc, c, n):
    return [tstar3(arc,c(t),tol,tol) for t in np.linspace(0,1,n)]


tol = 0.0001
TOL = 0.000001

x = lambda t: 0.5 + 0.3*t + 3.9*t**2 - 4.7*t**3
y = lambda t: 1.5 + 0.3*t + 0.9*t**2 - 2.7*t**3
dx = lambda t: 0.3 + 7.8*t - 14.1*t**2
dy = lambda t: 0.3 + 1.8*t - 8.1*t**2
arc = lambda t: np.sqrt(dx(t)**2 + dy(t)**2)
lTrap = adaptiveQuadrature(arc, 0, 1, TOL)
lSimp = adaptiveSimpsonsQuadrature(arc, 0, 1, TOL)
print("The length of the given curve is:")
print("Evaluated with the trapizoid method: "+str(lTrap))
print("Evaluated with the Simpsins method: "+str(lSimp))


s = 0.7
tstarr = tstar(arc, s, TOL, TOL)
print("The value of t*("+ str(s) + ") is: "+str(tstarr))


plotCurvePartition(x, y, equiPartition(arc, 4, tol), "Feril $P$ skipt í 4 hluta með helmingunarleit.")
plotCurvePartition(x, y, equiPartition(arc, 20, tol), "Feril $P$ skipt í 20 í hluta með helmingunarleit


tstarr2 = tstar2(arc, s, TOL, TOL)
print("The value of t*("+ str(s) + ") computed by Newton's method is: "+str(tstarr2))

plotCurvePartition(x, y, equiPartition2(arc, 4, tol), "Feril $P$ skipt í 4 hluta með aðferð Newtons.")
plotCurvePartition(x, y, equiPartition2(arc, 20, tol), "Feril $P$ skipt í 20 hluta með aðferð Newtons."


tstarr3 = tstar3(arc, s, TOL, TOL)
print("The value of t*("+ str(s) + ") computed by Newton's method is: "+str(tstarr3))
```

```
plotCurvePartition(x, y, equiPartition3(arc, 4, tol), "Feril $P$ skipt í 4 hluta með aðferð Newtons.")
plotCurvePartition(x, y, equiPartition3(arc, 20, tol), "Feril $P$ skipt í 20 hluta með aðferð Newtons.")

s=0.5
print("Time of computing t* with AQ trapizoid and bisection method:")
print(timeFunction(tstar, arc, s, tol, tol, number=10))

print("Time of computing t*2 with AQ trapizoid and Newtons method:")
print(timeFunction(tstar2, arc, s, tol, tol, number=10))

print("Time of computing t*3 with AQ Simpson and Newtons method:")
print(timeFunction(tstar3, arc, 0.5, tol, tol, number=10))

p1 = (0,0); p2 = (-1,1); p3 = (0,2); p4 = (0,1)
a1, a2, da1, da2 = Bezier(p1,p2,p3,p4)
arcBezier =  lambda t: np.sqrt(da1(t)**2 + da2(t)**2)
lBezSimp = adaptiveSimpsonsQuadrature(arcBezier, 0, 1, TOL)
print("The length of the Bézier curve is: "+ str(lBezSimp))

plotCurvePartition(a1, a2, equiPartition3(arcBezier, 4, tol),"Bézier ferlinum skipt í 4 hluta.")
plotCurvePartition(a1, a2, equiPartition3(arcBezier, 20, tol), "Bézier ferlinum skipt í 20 hluta.")

ani1 = animateCurve(x, y, np.linspace(0,1,200), "Ferillinn $P$ stikaður af $t$.")
ani1.save("ani1.mp4", writer="ffmpeg", fps=30)

sVec = [tstar3(arc,t,tol,tol) for t in np.linspace(0,1,200)]
ani2 = animateCurve(x, y, sVec, "Ferillinn $P$ stikaður af bogalengd.")
ani2.save("ani2.mp4", writer="ffmpeg", fps=30)

aniBez1 = animateCurve(a1, a2, np.linspace(0,1,200), "Bézier ferill stikaður af $t$.")
aniBez1.save("aniBez1.mp4", writer="ffmpeg", fps=30)

sVecBezier = [tstar3(arcBezier,t,tol,tol) for t in np.linspace(0,1,200)]
aniBez2 = animateCurve(a1, a2, sVecBezier, "Bézier ferill stikaður af bogalengd.")
aniBez2.save("aniBez2.mp4", writer="ffmpeg", fps=30)


ct1d3 = lambda t: np.power(t,1/3)
st1d3 = parametersFromProgressCurve(arc,ct1d3,300)
anit1d3 = animateCurve(x,y,st1d3, "Ferill $P$ umstikaður af $C(s)=s^\\frac{1}{3}$.")
anit1d3.save("anit1d3.mp4", writer="ffmpeg", fps=30)

ct2 = lambda t: np.power(t,2)
st2 = parametersFromProgressCurve(arc,ct2,150)
anit2 = animateCurve(x,y,st2, "Ferill $P$ umstikaður af $C(s)=s^2$.")
anit2.save("anit2.mp4", writer="ffmpeg", fps=30)

cSin1 = lambda t: np.sin(t*np.pi/2)
sSin1 = parametersFromProgressCurve(arc,cSin1,150)
aSin1 = animateCurve(x,y,sSin1, "Ferill $P$ umstikaður af $C(s)=\\sin(s\\pi /2)$.")
aSin1.save("anit2.mp4", writer="ffmpeg", fps=30)

cSin = lambda t: 1/2+1/2*np.sin((2*t-1)*np.pi/2)
sSin = parametersFromProgressCurve(arc,cSin,150)
```

```
aSin = animateCurve(x,y,sSin, "Ferill $P$ umstikaður af $C(s)=\\frac{1}{2}+\\frac{1}{2}\\sin((2s-1)\\pi/
aSin.save("aniSin.mp4", writer="ffmpeg", fps=30)

Hx = lambda t: 16*(np.sin(2*np.pi*t))**3
Hy = lambda t: 13*np.cos(2*np.pi*t)-5*np.cos(4*np.pi*t)-2*np.cos(6*np.pi*t)-np.cos(8*np.pi*t)
Hdx = lambda t: 48*((np.sin(2*np.pi*t))**2)*np.cos(2*np.pi*t)
Hdy = lambda t: -13*np.sin(2*np.pi*t)+10*np.sin(4*np.pi*t)+6*np.sin(6*np.pi*t)+4*np.cos(8*np.pi*t)
Harc = lambda t: np.sqrt(Hdx(2*np.pi*t)**2 + Hdy(2*np.pi*t)**2)

heart = animateCurve(Hx, Hy, np.linspace(0,1,200), "Hjarta, til gamans.")
heart.save("heart.mp4", writer="ffmpeg", fps=30)
```

Reykjavík, 25. mars 2019

Álfheiður Edda Sigurðardóttir

Hannes Kr. Árnason

Matthias Baldursson Harksen