

Aberystwyth University

A continuum description of the buckling of a line of spheres in a transverse harmonic confining potential

Hutzler, S.; Ryan-Purcell, J.; Mughal, A.; Weaire, D.

Published in:
Royal Society Open Science

DOI:
[10.1098/rsos.230293](https://doi.org/10.1098/rsos.230293)

Publication date:
2023

Citation for published version (APA):

Hutzler, S., Ryan-Purcell, J., Mughal, A., & Weaire, D. (2023). A continuum description of the buckling of a line of spheres in a transverse harmonic confining potential. *Royal Society Open Science*, 10(7), Article 230293. <https://doi.org/10.1098/rsos.230293>

Document License CC BY

General rights

Copyright and moral rights for the publications made accessible in the Aberystwyth Research Portal (the Institutional Repository) are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the Aberystwyth Research Portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the Aberystwyth Research Portal

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

tel: +44 1970 62 2400
email: is@aber.ac.uk

```

:=
In[ ]:= ClearAll["Global`*"]
(*PlotStyle section*)
text = {"Subtitle", 45, Bold};
options = {LabelStyle -> text,
  Axes -> False,
  Frame -> True,
  FrameStyle -> Thick,
  PlotRangePadding -> 0,
  AxesOrigin -> {0, 0},
  ImageSize -> 1000};

```

The 'Continuum Model'

Full Differential Equation - This is solved, for a given pair of $\{G, \tau\}$, by Mathematica's inbuilt NDSolve function. Helper functions are defined to implement a simple shooting method, designed to find the 'symmetric' solutions of interest that satisfy the boundary conditions.

```

In[ ]:= ClearAll[contSolNoShooting, endPointVal]
contSolNoShooting[g0_?NumericQ, p1_?NumericQ, t_:0, n_:10] :=
  Check[NDSolve[
    {
      
$$\phi''[u] == -4 \phi[u] + \frac{\phi[u]}{(g0 + t * u) \sqrt{1 + \phi[u]^2}}, \phi[0] == 0, \phi'[0] == p1$$
,
      
$$\phi, \{u, 0, n\}$$
 ], "Error"]

endPointVal[p_?NumericQ, g0_?NumericQ, t_?NumericQ] :=
  (
$$\phi[10] /. \text{contSolNoShooting}[g0, p, t, 10]$$
)[[1]]
(* We use this function to implement the shooting method
  for finding solutions to the differential equation *)

$$\Delta[\text{sol}_, n_?NumericQ] := \text{NIntegrate}[1 - \text{Cos}[\text{ArcTan}[\phi[u] /. \text{sol}], \{u, 0, n\}]$$

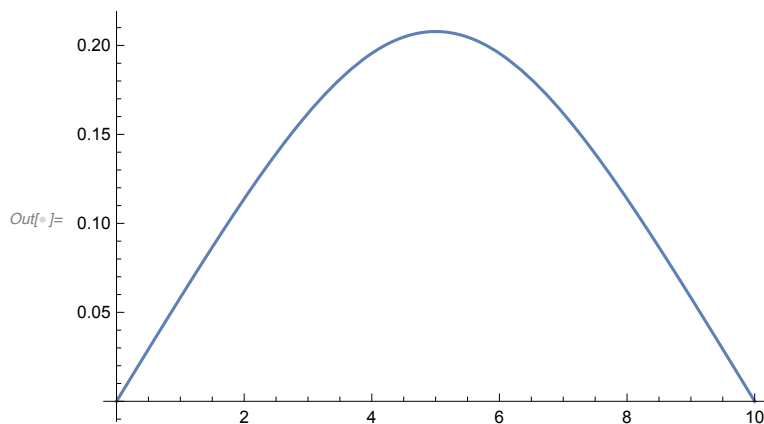
(* Calculating compression of a solution as discussed in the paper *)

(* For a given g0, t pair, find the correct value of the initial slope, p*)
slopeFind[g0_, t_, guess_] := FindRoot[endPointVal[p, g0, t],
  {p, guess}, DampingFactor -> 2, MaxIterations -> 500]

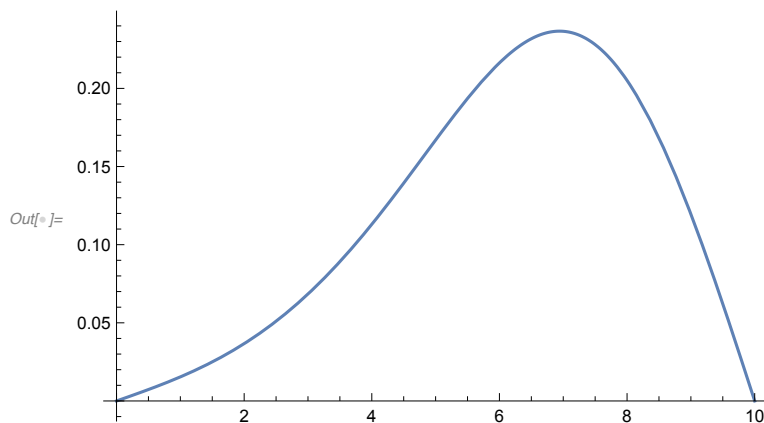
```

```
(* A sample calculation for N=10, t = 0*)
compressionG0[g0_?NumericQ, t_, s_] :=
Module[{slopeOld, slopeNew, c},
  slopeOld = s;
  slopeNew = p /. slopeFind[g0, t, slopeOld];
  c = Δ[contSolNoShooting[g0, slopeNew, t], 10];
  Return[{slopeNew, First[c]}]
]
compressionG0[0.252385, 0, 0.06];
slope = %[[1]];
Plot[Evaluate[
  φ[u] /. contSolNoShooting[0.252385, 0.0586601916835453`, 0, 10]], {u, 0, 10}]
```

Out[]:= {0.0586602, 0.100002}



```
In[ ]:= (* A sample calculation for N=10, t = 0.1*)
tilt = 0.005;
g0 = 0.22289;
compressionG0[g0, tilt, 0.01667676329462445`];
slope = %[[1]];
Plot[
  Evaluate[φ[u] /. contSolNoShooting[0.22289, 0.014326817348320982`, tilt, 10]],
  {u, 0, 10}]
```



Jacobi Solutions - Jacobi solutions are characterised by their $\{\kappa^2, m\}$ values and produce solutions with 0 tilt. These can be

determined by knowing the value of N and the target value of compression.

```
In[ ]:=  $\theta_{cn}[u_, k2s_, m_] := \text{ArcTan}\left[\sqrt{\frac{m * k2s}{\left(1 + \frac{k2s}{4}\right) (2m - 1)}}\right] * \text{JacobiCN}\left[\sqrt{\frac{k2s}{2m - 1}}(u), m\right];$ 
```

(* Note, this is defined in terms of θ rather than ϕ , hence the ArcTan *)

(*An example of a Jacobi solution,

with the red section highlighted showing the part of interest*)

```
m = 0.6
```

```
k = 0.031
```

```
Show[
```

```
Plot[ $\theta_{cn}[u - 5, k, m]$ , {u, 0, 10},
```

```
PlotStyle -> {{Red, Dashing[0.02], Thickness[0.01]}}],
```

```
Plot[ $\theta_{cn}[u - 5, k, m]$ , {u, 10, 30}, PlotStyle -> {{Thickness[0.01]}}],
```

```
Plot[ $\theta_{cn}[u - 5, k, m]$ , {u, -20, 0}, PlotStyle -> {{Thickness[0.01]}}],
```

```
Plot[0, {x, -20, 30}, PlotStyle -> Dashed],
```

```
options,
```

```
PlotRange -> Automatic,
```

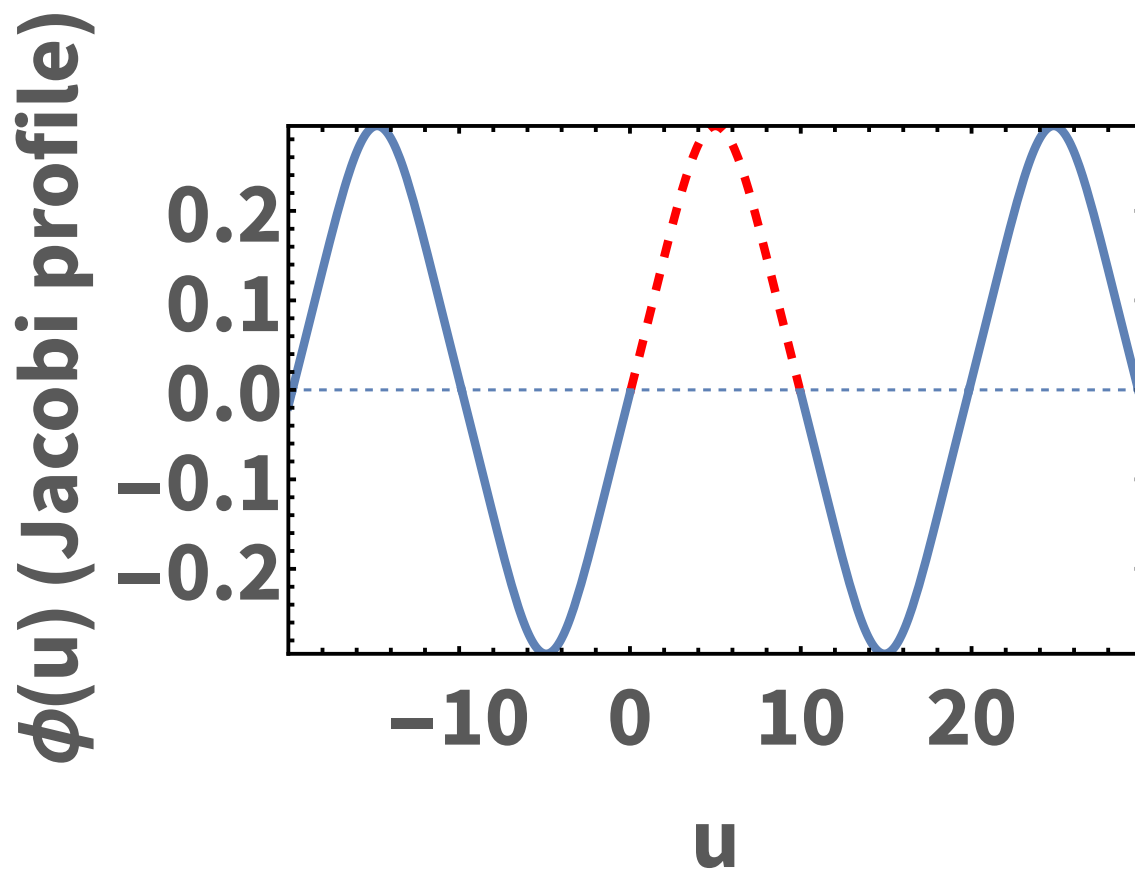
```
FrameLabel -> {"u", " $\phi(u)$  (Jacobi profile)"}
```

```
]
```

```
Out[ ]:= 0.6
```

```
Out[ ]:= 0.031
```

Out[]=



Airy Solutions - These are calculated using the rescaled variable x . After the rescaling, the solution can be written as a linear combination of Mathematica's inbuilt `AiryA` and `AiryB` functions. The target compression can be matched with a scaling variable c

```

In[ ]:= (* define the change of variables which turn
         the continuum equation into the Airy equation *)

x[u_, t_, g0_] :=  $\left(\frac{t}{g0^2}\right)^{-2/3} \left(\frac{1}{g0} \left(1 - \frac{t * u}{g0}\right) - 4\right)$ 

(* for a given t, g0, the roots of this Airy function need to
   be at (these calculations are done for N=10 exclusively): *)
x1[t_, g0_] := x[0, t, g0]
x2[t_, g0_] := x[10, t, g0]

y[x_, xroot_, c_:1] := Abs  $\left[ c * \left( \text{AiryAi}[x] - \frac{\text{AiryAi}[xroot]}{\text{AiryBi}[xroot]} * \text{AiryBi}[x] \right) \right]$ 

(* We take the absolute value as we
   are interested only in positive solutions *)
c1[d_?NumericQ, y_] :=
  c1 /. FindRoot[NIntegrate[1 - Cos[ArcTan[c1 * y]], {u, 0, 10}] == d, {c1, 1.7}]
(* This determines the prefactor c,
   which sets the compression of the solution *)

(* A sample Airy solution, showing the portion of interest in red *)

ttest = 0.001
g0test = 0.251
xtest[u_] := x[u, ttest, g0test]
x1test = x1[ttest, g0test]
x2test = x2[ttest, g0test]

Show[Plot[y[xtest[u], x1test, 1], {u, -7, 0}, PlotStyle -> {Thickness[0.01]}],
     Plot[y[xtest[u], x1test, 1], {u, 10, 40}, PlotStyle -> {Thickness[0.01]}],
     Plot[y[xtest[u], x2test, 1], {u, 0, 10},
          PlotStyle -> {Red, Dashing[0.02], Thickness[0.01]}],
     Plot[0, {x, -7, 40}, PlotStyle -> Dashed],
     PlotRange -> Automatic,
     options,
     FrameLabel -> {"u", " $\phi(u)$  (Airy profile)"}
]

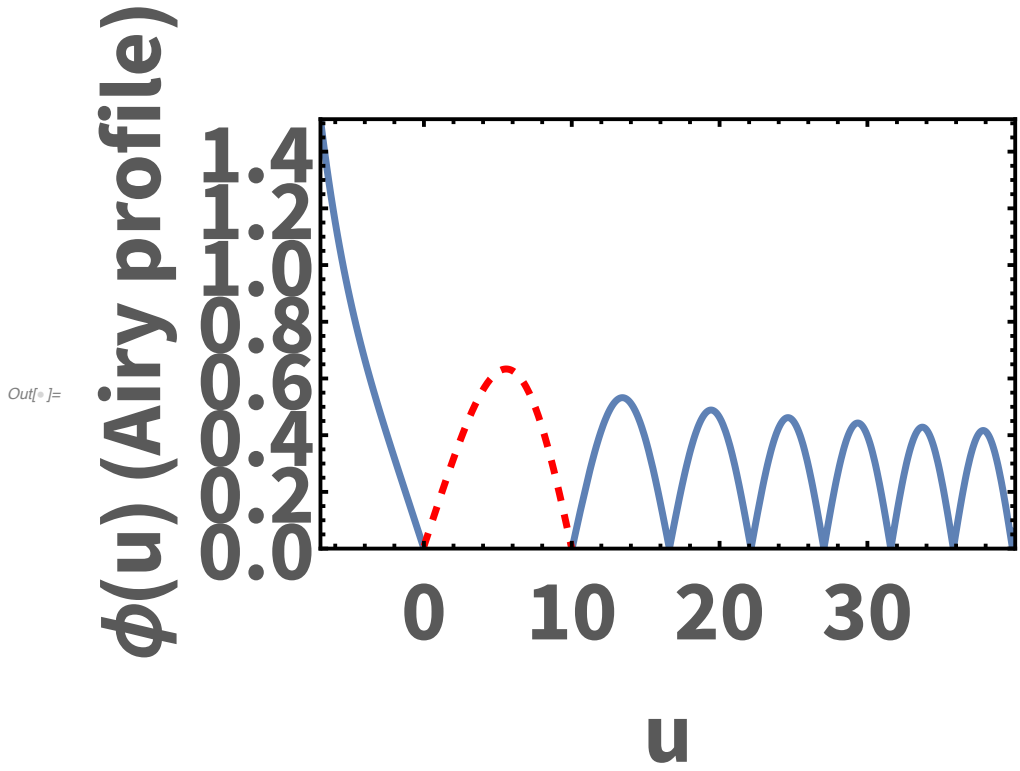
```

Out[]= 0.001

Out[]= 0.251

Out[]= -0.25232

Out[]= -2.76546



Whittaker Solutions - For a given value of N , the Whittaker solution has a corresponding value of τ , called τ_c . The compression of the solution is set by the value of the initial slope.

```

In[ ]:= ClearAll[solutionWhittaker]
solutionWhittaker[u_, λ_ : 1 * 10-10, tc_ : 0.0359113] :=
  λ / (4 i) WhittakerM[i / (4 * tc), 1 / 2, 4 i u] (* For N = 10, τc = 0.0359113 *)
ΔWhitt[λ_] := NIntegrate[1 - Cos[ArcTan[solutionWhittaker[u, λ]]], {u, 0, 10}]
(* The compression of a Whittaker solution
   is set only by the value of the initial slope *)
(* A sample Whittaker solution for N = 10,
   showing the part of initerest in red. *)
Show[
  Plot[Evaluate[solutionWhittaker[u]], {u, 0, 10},
    PlotStyle → {{Red, Dashing[0.02], Thickness[0.01]}}],
  Plot[Evaluate[solutionWhittaker[u]],
    {u, -10, 0}, PlotStyle → {Thickness[0.01]}],
  Plot[Evaluate[solutionWhittaker[u]],
    {u, 10, 20}, PlotStyle → {Thickness[0.01]}],
  Plot[0, {x, -10, 20}, PlotStyle → Dashed],
  options, FrameLabel → {"u", "ϕ(u) (Whittaker profile)"},
  PlotRange → Automatic
]

```