Error Formulae for Polynomial Collocation

Rolle's Theorem

Theorem

- *If* $f : [a, b] \to \mathbb{R}$ *continuous*
- *and* $f(x)$ *differentiable for* $a < x < b$
- *and* $f(a) = f(b)$

then there exists $c \in (a, b)$ *such that*

$$f'(c) = 0$$

Proof considers maximum of function f
applications in calculus

- proof of mean value theorem
- proof of Taylor's theorem

[https://en.wikipedia.org/wiki/Rolle%27s_theorem]

Theorem (Generalisation of Rolle's Theorem)

- *if* $f \in C^{n-1}[a, b]$
- *if* $f(x)$ *is* n *times differentiable for* $a < x < b$
- *if* $f(x_k) = c$ *for* $x_0 = a < x_1 < x_2 < \cdots < x_n = b$

then there exists $c \in (a, b)$ *such that*

$$f^{(n)}(c) = 0$$

Proof

- induction over n
- apply Rolle's theorem to subintervals $[x_k, x_{k+1}]$

Error Formula

Theorem (Error of Lagrangian interpolation)

- *if* p_n *polynomial (Lagrangian) interpolant of* $f(x) \in C^{(n+1)}[a, b]$
- *and* $x_i, x \in [a, b]$ *distinct real numbers,* $i = 0, \ldots, n$

then the interpolating polynomial p of degree n satisfies

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)\, w(x), \quad \text{for some } \xi \in [a, b]$$

where $w(x) = (x - x_0) \cdots (x - x_n)$

Proof

- $G(t) := (f(x) - p(x))\,\omega(t) - (f(t) - p(t))\,\omega(x)$
- then $G(x_i) = G(x) = 0$, Rolle: $G^{(n+1)}(\xi) = 0$ for some ξ
- $G^{(n+1)}(t) = (f(x) - p(x))(n+1)! - f^{(n+1)}(t)\,\omega(x)$

D.A. Arnold, ‘A Concise Introduction to Numerical Analysis’, 2001, [http://www.ima.umn.edu/∼arnold]

Example: error of interpolating $f(x) = \sin(x)$ on $[0, 1]$

- all derivatives f are either $\pm\sin(x)$ or $\pm\cos(x)$ and it follows that

$$|f^{(k)}(x)| \leq 1$$

- for $x, x_k \in [0, 1]$ one has

$$|w(x)| \leq 1$$

consequently, in this case the error is bounded by

$$|f(x) - p(x)| = \frac{1}{(n+1)!}$$

- if $n = 9$, i.e. 10 nodes then

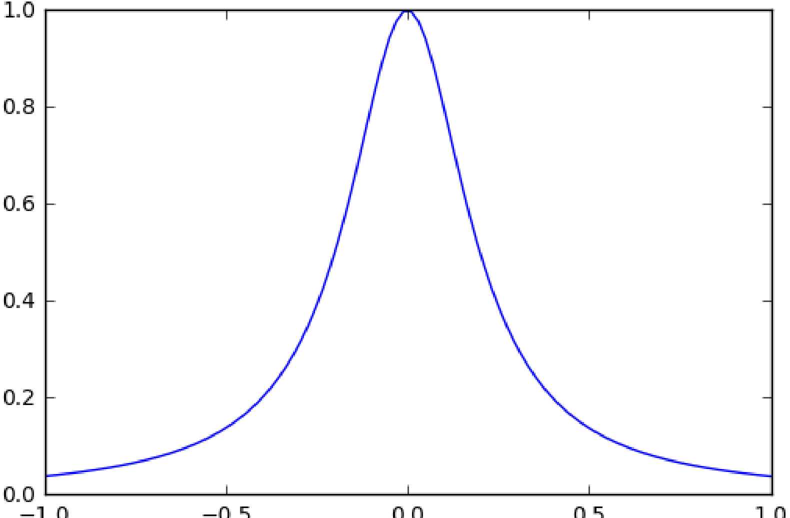
$$|\sin x - p(x)| \leq \frac{1}{10!} < 2.8 \times 10^{-7}$$

Comment

- it is seen that often the error close to a multiple of $w(x)$ and thus choosing x_k to achieve small $|w(x)|$ is a good strategy

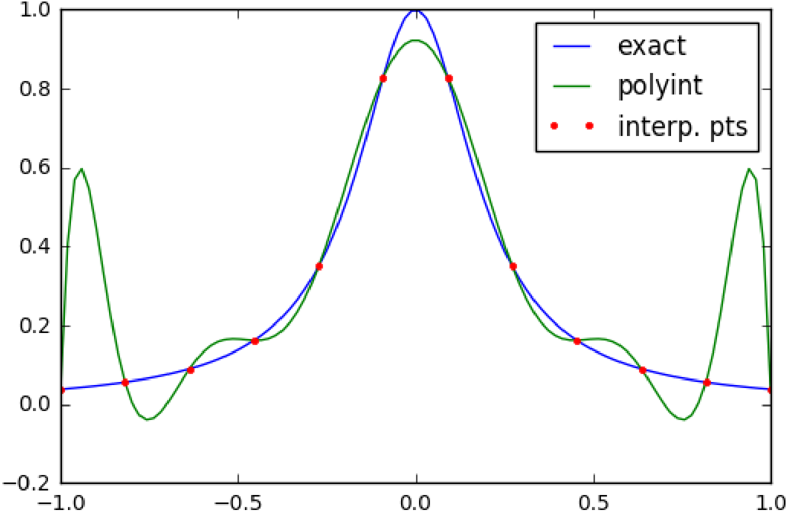
Example: Runge's function $f(t) = \frac{1}{1+25\,t^2}$

```
fr = lambda t : 1.0/(1.0+25*t**2)
ng = 100; xg = np.linspace(-1.0,1.0,ng)
plt.plot(xg, fr(xg),'-');
```



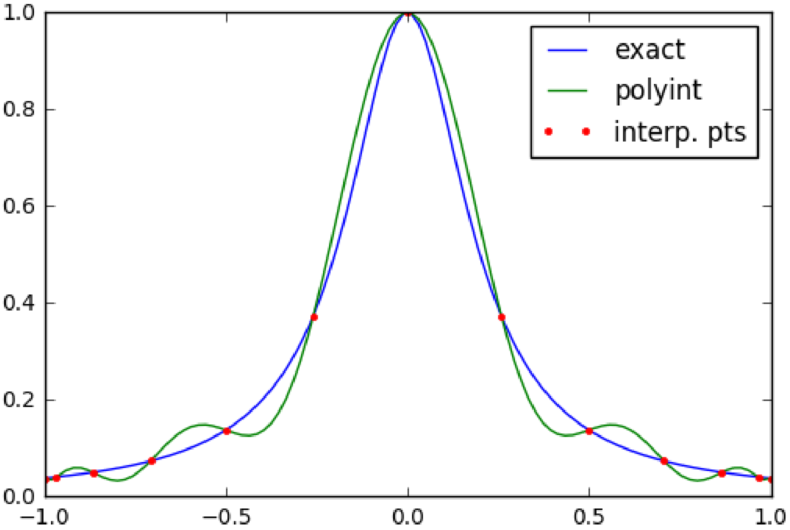
```python
# stable polynomial interpolation for Runge example
# barycentric method and equidistant points

fr = lambda t : 1.0/(1.0+25*t**2)      # Runge function

ng = 12
xg = np.linspace(-1.0,1.0,ng)

pr = spyint.BarycentricInterpolator(xg, fr(xg))

xplt = np.linspace(-1.0, 1.0, 100)
```

```
plt.plot(xplt, fr(xplt),'-', label='exact')
plt.plot(xplt, pr(xplt),'-', label='polyint')
plt.plot(xg, pr(xg),'.', label='interp. pts')
plt.legend();
```



```
# stable polynomial interpolation for Runge example using
# barycentric method and Chebyshev points (see section bel
fr = lambda t : 1.0/(1.0+25*t**2)     # Runge function
ng = 13
xg = np.cos(np.linspace(0.0, np.pi,ng))    # Chebyshev poi
pr = BarycentricInterpolator(xg, fr(xg))
xplt = np.linspace(-1.0, 1.0, 100)
```

```
plt.plot(xplt, fr(xplt),'-', label='exact')
plt.plot(xplt, pr(xplt),'-', label='polyint')
plt.plot(xg, pr(xg),'.', label='interp. pts')
plt.legend();
```



Bernstein approximation – an alternative to interpolation

- Bernstein polynomials

$$b_{k,n}(x) = \binom{n}{k} x^k (1-x)^{n-k}, \quad k = 0, \ldots, n, x \in [0,1]$$

- Bernstein approximation of $f(x)$

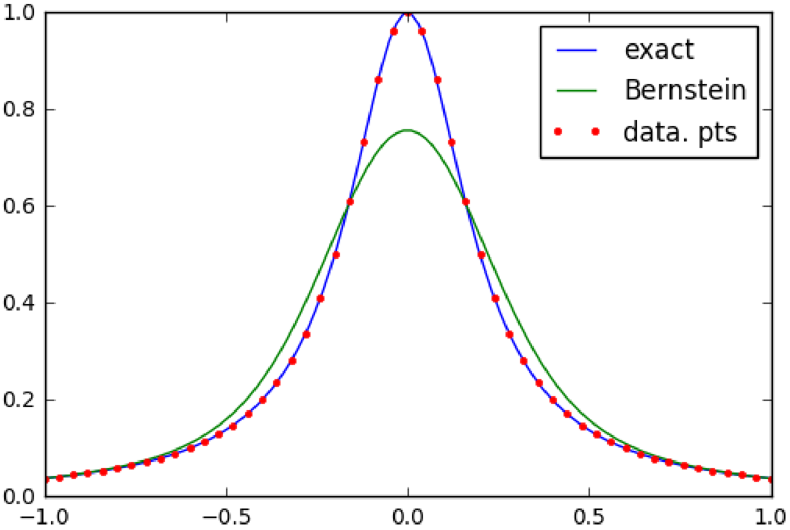
$$p(x) = \sum_{k=0}^{n} f(k/n) b_{k,n}(x)$$

- can show that interpolant converges uniformly for continuous f
  - proof of Weierstrass theorem
- probabilistic interpretation:
  - $b_{k,n}(x)$ binomial probability of choosing k given n and $x \in [0,1]$
  - Bernstein approximation is then the expectation of $f(k/n)$
  - approximate by sampling

[https://en.wikipedia.org/wiki/Bernstein_polynomial]

```python
# Bernstein approximation from scipy

fr = lambda t : 1.0/(1.0+25*t**2)      # Runge function

ng = 51
xg = np.linspace(-1.0, 1.0, ng)
yg = fr(xg)
yg.shape = yg.shape+(1,)
pr = BPoly(yg, [0,1])

xplt = np.linspace(-1.0, 1.0, 100)
```

```
plt.plot(xplt, fr(xplt),'-', label='exact')
plt.plot(xplt, pr((xplt+1)/2),'-', label='Bernstein')
plt.plot(xg, fr(xg),'.', label='data. pts')
plt.legend();
```



```python
# simulation using samples from binomial distribution
from numpy.random import binomial

fr = lambda t : 1.0/(1.0+25*t**2)      # Runge function

# data points (random variable)
ng = 51    # data size
xg = np.linspace(-1.0, 1.0, ng)
yg = fr(xg)

nplt = 200     # evaluation grid
xplt = np.linspace(-1.0,1.0,nplt)
yplt = np.zeros(nplt)

ns = 100    # sample size
for k in range(nplt):
    i = binomial(ng-1, (xplt[k]+1)/2.0,ns)
    yplt[k] = np.mean(yg[i])
```

```
plt.plot(xplt, fr(xplt),'-', label='exact')
plt.plot(xplt, yplt, '+', label='sample approx.')
plt.plot(xg, fr(xg),'.', label='data. pts')
plt.legend();
```