NumeriKet

Version 6.5

May 30, 2016

A collection of Racket modules implementing numerical methods.

1 Overview

Basic numerical methods, such as

- Newton's method for square roots
- Newton's method for roots of polynomials
- Euler's method for solving ordinary differential equations

are included.

```
#lang racket/base
(require NumeriKet)

(newton-sqrt 4 4)
(newton-root (lambda (x) (- (sin x) (cos x) -1)) -1)
```

```
> racket test.rkt
2.0
-1.5707963267948966
```

2 Implemented Functions

2.1 Root Finding

```
(newton-root f x0) → number?
  f : procedure?
  x0 : number?
```

Approximate the root of `f` closest to `x0`.

This is implemented through Newton's method for finding roots of real valued functions, and uses ten iterations. Note that `f` must have a continuous second derivative.

```
(newton-sqrt s x0) → number?
  s : number?
  x0 : number?
```

Approximate the square root of `s` by starting at `x0`.

This is implemented through a specific case of `newton-root` and uses ten iterations.

2.2 Derivatives

```
(diff f x0) → number?
  f : procedure?
  x0 : number?
```

Estimate the value of the derivative of `f`, a function of `x`, at the point `x0`.

This is implemented through Newton's difference quotient with a step size of 1e-7.

2.3 Solving ODEs

```
(euler-method f inits tf) → number?
  f : procedure?
  inits : list?
  tf : number?
```

Given `(define t0 (first inits))` and `(define x0 (second inits))`, approximate the value of the solution of `f` at `tf`, where `(x t0)` is `x0`.

This is implemented through Euler's method for solving first order differential equations, and uses a step size of 0.0001. Note that `f`, the flow of `x` in time, is a function of `t` and `x` (in that order).

2.4 Utility Functions

```
(round-to-precision r p) → number?
  r : number?
  p : number?
```

Round the value `r` to `p` decimal places.